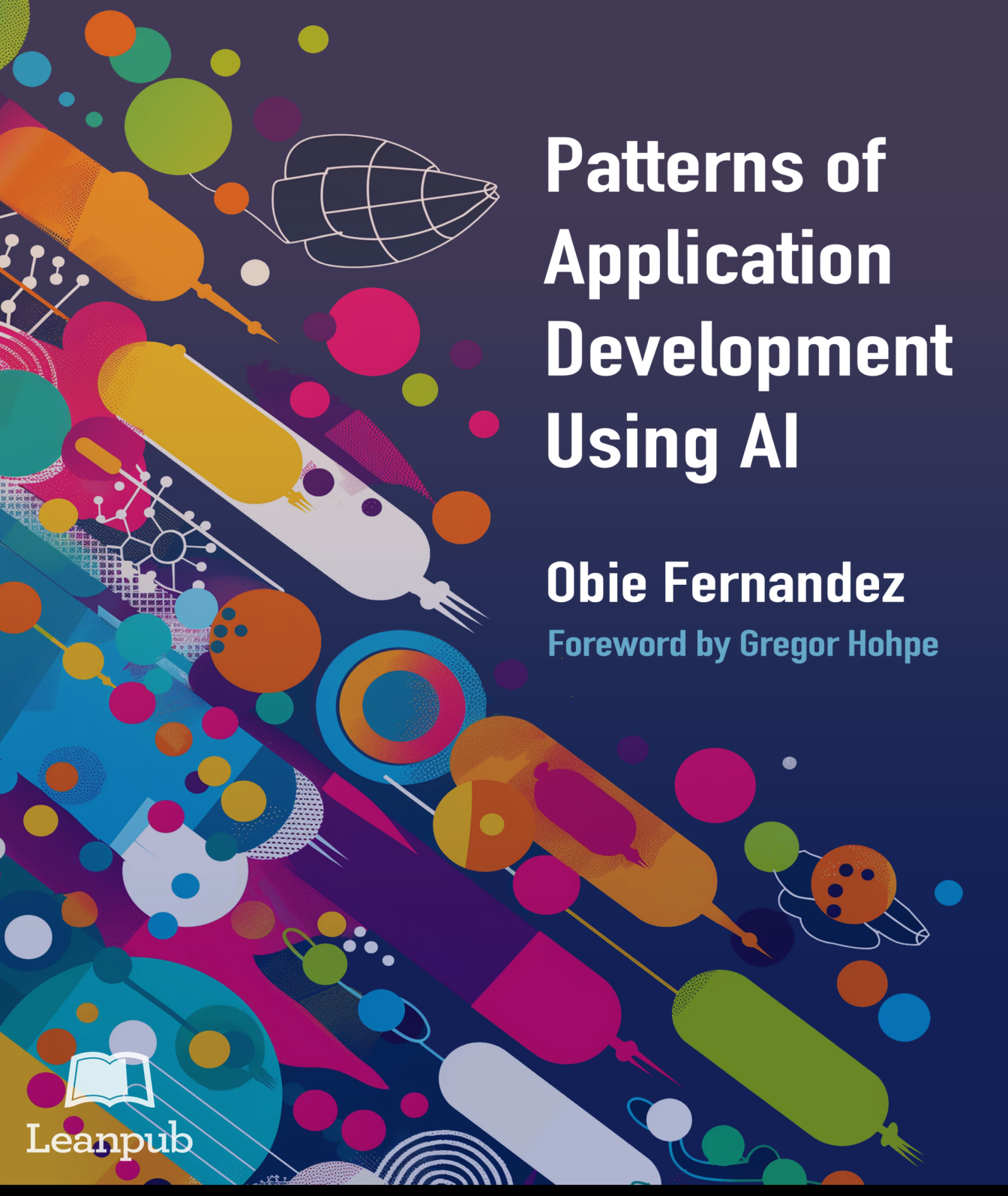


מהדורה בעברית

דפוסי פיתוח יישומים באמצעות בינה מלאכותית (מהדורה עברית)

Fernandez Obie

הספר הזה מוצע למכירה ב:

http://leanpub.com/patterns-of-application-development-using-ai-he

גרסא זו פורסמה בתאריך 2025-01-23

צייצו על הספר הזה!

אנא עזרו לFernandez Obie בכך שתפיצו את השמועה אודות הספר הזה ב טוויטר!

ההאשטאג המוצע לספר זה הוא #poaduai.

גלו מה אנשים אחרים אומרים על הספר על ידי לחיצה על קישור זה לחיפוש ההאשטאג הזה בטוויטר:

#poaduai

לויקטוריה, מלכתי המטורפת, המוזה שלי, האור והאהבה שלי

Fernandez Obie עוד מאת

AI Using Development Application of Patterns

Way 8 Rails The

Way 7 Rails The

Way Rails The XML

Serverless

Node de Principiante Libro El

Enterprise Lean The

תוכן העניינים

הקדמה מאת Hohpe Gregor

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

הקדמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

אודות הספר

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

אודות דוגמאות הקוד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

במה איני עוסק

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

למי מיועד הספר הזה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

בניית אוצר מילים משותף

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

להיות מעורב

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

תודות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מה עם האיורים?

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

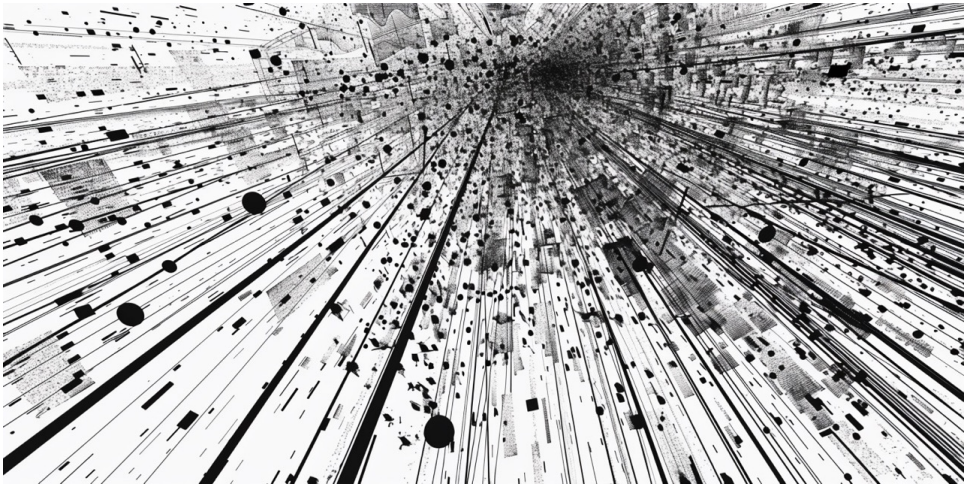
אודות פרסום רזה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

אודות המחבר

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מבוא



אם אתם להוטים להתחיל לשלב מודלי שפה גדולים של בינה מלאכותית (LLMs) בפרויקטים התכנותיים שלכם, אתם מוזמנים לצלול ישירות לתבניות ודוגמאות הקוד המוצגות בפרקים הבאים. עם זאת, כדי להעריך במלואה את העוצמה והפוטנציאל של תבניות אלה, כדאי לקחת רגע להבין את ההקשר הרחב יותר ואת הגישה המאוחדת שהן מייצגות.

התבניות אינן רק אוסף של טכניקות מבודדות אלא מסגרת עבודה מאוחדת לשילוב בינה מלאכותית באפליקציות שלכם. אני משתמש ב-Rails on Ruby, אך תבניות אלו אמורות לעבוד כמעט בכל סביבת תכנות אחרת. הן מתייחסות למגוון רחב של סוגיות, מניהול נתונים ואופטימיזציית ביצועים ועד לחוויית משתמש ואבטחה, ומספקות ארגז כלים מקיף להעשרת שיטות התכנות המסורתיות עם יכולות בינה מלאכותית.

כל קטגוריה של תבניות מתמודדת עם אתגר או הזדמנות ספציפיים שעולים בעת שילוב רכיבי בינה מלאכותית באפליקציה שלכם. על ידי הבנת הקשרים והסינרגיות בין תבניות אלו, תוכלו לקבל החלטות מושכלות לגבי היכן וכיצד ליישם בינה מלאכותית באופן היעיל ביותר.

תבניות אינן פתרונות מרשמיים ואין להתייחס אליהן ככאלה. הן נועדו להיות אבני בניין גמישות שיש להתאים לדרישות ולאילוצים הייחודיים של האפליקציה הייחודית שלכם. היישום המוצלח של תבניות אלו (כמו כל תבנית אחרת בתחום התוכנה) מסתמך על הבנה עמוקה של תחום הבעיה, צרכי המשתמש והארכיטקטורה הטכנית הכוללת של הפרויקט שלכם.

מחשבות על ארכיטקטורת תוכנה

התחלתי לתכנת בשנות ה-80 והייתי מעורב בסצנת ההאקרים, ומעולם לא איבדתי את הגישה ההאקרית שלי, גם אחרי שהפכתי למפתח תוכנה מקצועי. מההתחלה, תמיד הייתה לי ספקנות בריאה לגבי הערך שארכיטקטי תוכנה במגדלי השן שלהם באמת הביאו לשולחן.

אחת הסיבות שאני באופן אישי כל כך נרגש מהשינויים שמביא איתו גל הטכנולוגיה החדש והעוצמתי הזה של בינה מלאכותית היא ההשפעה שלו על מה שאנחנו מחשיבים כהחלטות ארכיטקטורת תוכנה. הוא מאתגר תפיסות מסורתיות של מה מהווה את הדרך ה"נכונה" לתכנן וליישם את פרויקטי התוכנה שלנו. הוא גם מאתגר האם ניתן עדיין לחשוב על ארכיטקטורה בעיקר כהחלטים במערכת שקשה לשנות, מכיוון שהעצמה באמצעות בינה מלאכותית הופכת את זה לקל יותר מתמיד לשנות כל חלק בפרויקט שלכם, בכל זמן.

ייתכן שאנחנו נכנסים לשנות השיא של הגישה ה"פוסט-מודרנית" להנדסת תוכנה. בהקשר זה, פוסט-מודרני מתייחס לשינוי יסודי המתרחק מהפרדיגמות המסורתיות, שבהן מפתחים היו אחראים לכתיבה ותחזוקה של כל שורת קוד. במקום זאת, היא מאמצת את הרעיון של האצלת משימות, כגון מניפולציה של נתונים, אלגוריתמים מורכבים, ואפילו חלקים שלמים של לוגיקה אפליקטיבית, לספריות צד שלישי וממשקי תכנות חיצוניים. שינוי פוסט-מודרני זה מייצג סטייה משמעותית מהחוכמה המקובלת של בניית אפליקציות מהיסוד, והוא מאתגר מפתחים לחשוב מחדש על תפקידם בתהליך הפיתוח.

תמיד האמנתי שמתכנתים טובים כותבים רק את הקוד שהכרחי לכתוב, בהתבסס על תורתו

של לארי וול והאקרים מוארים אחרים כמותו. על ידי מזעור כמות הקוד שנכתב, אנחנו יכולים לנוע מהר יותר, להפחית את שטח הפנים לבאגים, לפשט את התחזוקה, ולשפר את האמינות הכוללת של האפליקציות שלנו. פחות קוד מאפשר לנו להתמקד בלוגיקה העסקית הליבתית ובחוויית המשתמש, תוך האצלת עבודה אחרת לשירותים אחרים.

כעת, כשמערכות מבוססות בינה מלאכותית יכולות לטפל במשימות שהיו בעבר נחלתו הבלעדית של קוד שנכתב על ידי בני אדם, אנחנו אמורים להיות מסוגלים להיות יצרניים וזריזים אף יותר, עם מיקוד גדול מתמיד ביצירת ערך עסקי וחוויית משתמש.

כמובן שישנם חסרונות בהאצלת חלקים גדולים מהפרויקט שלך למערכות בינה מלאכותית, כגון אובדן פוטנציאלי של שליטה, והצורך במנגנוני ניטור ומשוב חזקים. זו הסיבה שזה דורש מערכת חדשה של כישורים וידע, כולל לפחות הבנה בסיסית של אופן פעולת הבינה המלאכותית.

מהו מודל שפה גדול?

מודלי שפה גדולים (LLMs) הם סוג של מודל בינה מלאכותית שזכו לתשומת לב משמעותית בשנים האחרונות, מאז השקת GPT-3 על ידי OpenAI בשנת 2020. מודלי שפה גדולים מתוכננים לעבד, להבין וליצור שפה אנושית בדיוק ושטף מרשימים. בחלק זה, נעיף מבט קצר על אופן פעולתם של מודלי שפה גדולים ומדוע הם מתאימים לבניית רכיבי מערכת חכמים.

בליבם, מודלי שפה גדולים מבוססים על אלגוריתמי למידה עמוקה, ובפרט רשתות עצביות. רשתות אלה מורכבות מצמתים מקושרים, או נוירונים, שמעבדים ומשדרים מידע. הארכיטקטורה הנבחרת עבור מודלי שפה גדולים היא לעתים קרובות מודל הטרנספורמר, שהוכיח את עצמו כיעיל במיוחד בטיפול בנתונים רציפים כמו טקסט.

מודלי טרנספורמר מבוססים על מנגנון קשב ומשמשים בעיקר למשימות הכוללות נתונים רציפים, כמו עיבוד שפה טבעית. טרנספורמרים מעבדים את נתוני הקלט בבת אחת במקום באופן רציף, מה שמאפשר להם ללכוד תלויות ארוכות טווח ביעילות רבה יותר. הם מכילים שכבות של מנגנוני קשב המסייעים למודל להתמקד בחלקים שונים של נתוני הקלט כדי להבין הקשר ויחסים.

תהליך האימון של מודלי שפה גדולים כולל חשיפת המודל לכמויות עצומות של נתוני טקסט, כגון ספרים, מאמרים, אתרי אינטרנט ומאגרי קוד. במהלך האימון, המודל לומד לזהות דפוסים, יחסים ומבנים בתוך הטקסט. הוא לוכד את המאפיינים הסטטיסטיים של השפה, כגון כללי דקדוק, קשרים בין מילים ומשמעויות הקשריות.

אחת הטכניקות המרכזיות המשמשות באימון מודלי שפה גדולים היא למידה לא מפוקחת. משמעות הדבר היא שהמודל לומד מהנתונים ללא תיוג או הכוונה מפורשים. הוא מגלה דפוסים וייצוגים בעצמו על ידי ניתוח ההופעה המשותפת של מילים וביטויים בנתוני האימון. זה מאפשר למודלי שפה גדולים לפתח הבנה עמוקה של השפה ומורכבויותיה.

היבט חשוב נוסף של מודלי שפה גדולים הוא יכולתם לטפל בהקשר. בעת עיבוד טקסט, מודלי שפה גדולים מתחשבים לא רק במילים הבודדות אלא גם בהקשר הסובב. הם לוקחים בחשבון את המילים, המשפטים ואפילו הפסקאות הקודמות כדי להבין את המשמעות והכוונה של הטקסט. הבנת ההקשר הזו מאפשרת למודלי שפה גדולים לייצר תגובות קוהרנטיות ורלוונטיות. אחת הדרכים העיקריות שבהן אנו מעריכים את היכולות של מודל שפה גדול מסוים היא על ידי בחינת גודל ההקשר שהם יכולים לקחת בחשבון כדי לייצר תגובות.

לאחר האימון, ניתן להשתמש במודלי שפה גדולים למגוון רחב של משימות הקשורות לשפה. הם יכולים לייצר טקסט דמוי-אנושי, לענות על שאלות, לסכם מסמכים, לתרגם שפות ואפילו לכתוב קוד. הרב-גוניות של מודלי שפה גדולים הופכת אותם לבעלי ערך בבניית רכיבי מערכת חכמים שיכולים לתקשר עם משתמשים, לעבד ולנתח נתוני טקסט, ולייצר פלטים משמעותיים.

על ידי שילוב מודלי שפה גדולים בארכיטקטורת היישום, ניתן ליצור רכיבי בינה מלאכותית המבינים ומעבדים קלט משתמש, מייצרים תוכן דינמי ומספקים המלצות או פעולות חכמות. אך עבודה עם מודלי שפה גדולים דורשת התחשבות זהירה בדרישות משאבים ופשרות ביצועים. מודלי שפה גדולים צורכים משאבי מחשוב רבים ועשויים לדרוש כוח עיבוד וזיכרון משמעותיים (במילים אחרות, כסף) כדי לפעול. רובנו נצטרך להעריך את השלכות העלות של שילוב מודלי שפה גדולים ביישומים שלנו ולפעול בהתאם.

הבנת הסקה

הסקה מתייחסת לתהליך שבו מודל מייצר תחזיות או פלט על בסיס נתונים חדשים שלא ראה מעולם. זהו השלב שבו המודל המאומן משמש לקבלת החלטות או ליצירת טקסט, תמונות או תוכן אחר בתגובה לקלט המשתמש.

במהלך שלב האימון, מודל בינה מלאכותית לומד ממסד נתונים גדול על ידי התאמת הפרמטרים שלו כדי למזער את השגיאה בתחזיותיו. לאחר האימון, המודל יכול ליישם את מה שלמד על נתונים חדשים. הסקה היא האופן שבו המודל משתמש בדפוסים ובידע שלמד כדי לייצר פלט.

עבור מודלים שפתיים גדולים, הסקה כוללת לקיחת פרומפט או טקסט קלט וייצור תגובה קוהרנטית ורלוונטית להקשר, כזרם של אסימונים (שעליהם נדבר בקרוב). זה יכול להיות מענה על שאלה, השלמת משפט, יצירת סיפור, או תרגום טקסט, בין משימות רבות אחרות.

בניגוד לאופן שבו אתה ואני חושבים, ה"חשיבה" של מודל בינה מלאכותית דרך הסקה מתרחשת כולה בפעולה חסרת מצב אחת. כלומר, החשיבה שלו מוגבלת לתהליך היצירה שלו. הוא ממש חייב לחשוב בקול רם, כאילו שאלתי אותך שאלה וקיבלתי ממך תשובה רק בסגנון "זרם תודעה".

מודלים שפתיים גדולים מגיעים במגוון גדלים וטעמים

בעוד שכמעט כל המודלים השפתיים הגדולים (LLMs) הפופולריים מבוססים על אותה ארכיטקטורת טרנספורמר בסיסית ומאומנים על מאגרי טקסט ענקיים, הם מגיעים במגוון גדלים ומכוונים למטרות שונות. גודל המודל השפתי, הנמדד במספר הפרמטרים ברשת העצבית שלו, משפיע מאוד על יכולותיו. מודלים גדולים יותר עם יותר פרמטרים, כמו GPT-4, שלפי השמועות מתהדר ב-1 עד 2 טריליון פרמטרים, הם בדרך כלל בעלי ידע ויכולות רבות יותר ממודלים קטנים יותר. עם זאת, מודלים גדולים יותר דורשים הרבה יותר כוח מחשוב להפעלה, מה שמתורגם לעלות גבוהה יותר כשמשתמשים בהם דרך קריאות API.

כדי להפוך את המודלים השפתיים הגדולים למעשיים יותר ומותאמים למקרי שימוש ספציפיים, המודלים הבסיסיים עוברים לעתים קרובות כוונון עדין על מאגרי נתונים ממוקדים יותר.

למשל, מודל שפתי גדול עשוי להיות מאומן על קורפוס גדול של דיאלוגים כדי להתמחות בבינה מלאכותית שיחתית. אחרים מאומנים על קוד כדי להעניק להם ידע בתכנות. יש אפילו מודלים שאומנו במיוחד לאינטראקציות בסגנון משחקי תפקידים עם משתמשים!

מודלים מבוססי-אחזור לעומת מודלים גנרטיביים

בעולם המודלים השפתיים הגדולים (LLMs), ישנן שתי גישות עיקריות ליצירת תשובות: מודלים מבוססי-אחזור ומודלים גנרטיביים. לכל גישה יש את היתרונות והחסרונות שלה, והבנת ההבדלים ביניהן יכולה לעזור לך לבחור את המודל המתאים למקרה השימוש הספציפי שלך.

מודלים מבוססי-אחזור

מודלים מבוססי-אחזור, הידועים גם כמודלים לאחזור מידע, מייצרים תשובות על ידי חיפוש במסד נתונים גדול של טקסטים קיימים ובחירת הקטעים הרלוונטיים ביותר בהתבסס על שאילתת הקלט. מודלים אלה אינם מייצרים טקסט חדש מאפס אלא מחברים קטעים מתוך מסד הנתונים ליצירת תשובה קוהרנטית.

אחד היתרונות העיקריים של מודלים מבוססי-אחזור הוא יכולתם לספק מידע מדויק ועדכני. מכיוון שהם מסתמכים על מסד נתונים של טקסטים מאורגנים, הם יכולים לשלוף מידע רלוונטי ממקורות אמינים ולהציגו למשתמש. זה הופך אותם למתאימים במיוחד ליישומים הדורשים תשובות מדויקות ועובדתיות, כמו מערכות שאלות-תשובות או מאגרי ידע.

עם זאת, למודלים מבוססי-אחזור יש מספר מגבלות. הם טובים רק כמו מסד הנתונים שבו הם מחפשים, כך שאיכות וכיסוי מסד הנתונים משפיעים ישירות על ביצועי המודל. בנוסף, מודלים אלה עשויים להתקשות ביצירת תשובות קוהרנטיות וטבעיות, מכיוון שהם מוגבלים לטקסט הזמין במסד הנתונים.

איננו מכסים שימוש במודלי אחזור טהורים בספר זה.

מודלים גנרטיביים

מודלים גנרטיביים, לעומת זאת, יוצרים טקסט חדש מאפס בהתבסס על הדפוסים והקשרים שהם למדו במהלך האימון. מודלים אלה משתמשים בהבנת השפה שלהם כדי ליצור תשובות חדשות המותאמות לפקודת ההפעלה.

החוזק העיקרי של מודלים גנרטיביים הוא יכולתם ליצור טקסט יצירתי, קוהרנטי ורלוונטי להקשר. הם יכולים לנהל שיחות פתוחות, ליצור סיפורים, ואפילו לכתוב קוד. זה הופך אותם לאידיאליים ליישומים הדורשים אינטראקציות פתוחות ודינמיות יותר, כמו צ'אטבוטים, יצירת תוכן ועוזרי כתיבה יצירתית.

עם זאת, מודלים גנרטיביים עלולים לעתים לייצר מידע לא עקבי או שגוי מבחינה עובדתית, מכיוון שהם מסתמכים על הדפוסים שנלמדו במהלך האימון ולא על מסד נתונים מאורגן של עובדות. הם עשויים גם להיות נוטים יותר להטיות ולהזיות, ולייצר טקסט שנשמע סביר אך אינו בהכרח אמיתי.

דוגמאות למודלי LLM יוצרים כוללות את סדרת ה-GPT של OpenAI (GPT-3, GPT-4) ואת Claude של Anthropic.

מודלים היברידיים

מספר מודלי LLM מסחריים משלבים גישות של אחזור ויצירה במודל היברידי. מודלים אלה משתמשים בטכניקות אחזור כדי למצוא מידע רלוונטי ממסד נתונים ולאחר מכן משתמשים בטכניקות יצירה כדי לסנתז את המידע הזה לתשובה קוהרנטית.

מודלים היברידיים שואפים לשלב את הדיוק העובדתי של מודלים מבוססי אחזור עם יכולות יצירת השפה הטבעית של מודלים יוצרים. הם יכולים לספק מידע מהימן ועדכני יותר תוך שמירה על היכולת לנהל שיחות פתוחות.

בבחירה בין מודלים מבוססי אחזור למודלים יוצרים, עליך לשקול את הדרישות הספציפיות של היישום שלך. אם המטרה העיקרית היא לספק מידע עובדתי מדויק, מודל מבוסס אחזור עשוי להיות הבחירה הטובה ביותר. אם היישום דורש אינטראקציות פתוחות ויצירתיות יותר, מודל יוצר עשוי להיות מתאים יותר. מודלים היברידיים מציעים איזון בין שתי הגישות ויכולים להיות בחירה טובה ליישומים הדורשים הן דיוק עובדתי והן יצירת שפה טבעית.

בסופו של דבר, הבחירה בין מודלים מבוססי אחזור למודלים יוצרים תלויה במקרה השימוש הספציפי ובאיזון בין דיוק, יצירתיות וגמישות. על ידי הבנת החוזקות והמגבלות של כל גישה, תוכל לקבל החלטות מושכלות.

כוונון הוראות

כוונון הוראות הוא תת-קבוצה של אימון שיחתי המשתמש בהנחיות ותשובות שנכתבו על ידי בני אדם כדי לשפר את יכולת ה-LLM להבין ולבצע הוראות. הנה כמה דוגמאות פשוטות להמחשת כיצד כוונון הוראות על הנחיות ותשובות שנכתבו על ידי בני אדם משפר את יכולת ה-LLM להבין ולבצע הוראות:

דוגמה 1: תמיכת לקוחות

הוראה: "ספק תשובה ללקוח המבקש החזר כספי."

הנחיה שנכתבה על ידי אדם: "ברצוני לבקש החזר כספי עבור הרכישה האחרונה שלי. המוצר לא עמד בציפיות שלי."

תשובה שנכתבה על ידי אדם: "אני מצטער לשמוע שהמוצר לא עמד בציפיות שלך. כדי לעבד את ההחזר הכספי שלך, אנא ספק את מספר ההזמנה שלך ואת הסיבה להחזרה. נבחן את בקשתך ונחזור אליך תוך 3-2 ימי עסקים."

דוגמה 2: סיוע חינוכי

הוראה: "הסבר את מושג הפוטוסינתזה לילד בן 10."

שאלה בכתיבה אנושית: "האם תוכל להסביר כיצד צמחים מייצרים את מזונם?"

תשובה בכתיבה אנושית: "בוודאי! צמחים מייצרים את מזונם באמצעות תהליך הנקרא פוטוסינתזה. הם משתמשים באור השמש, מים ופחמן דו-חמצני כדי ליצור מזון וחמצן. בעלי הצמח יש חלק מיוחד הנקרא כלורופיל שעוזר להם לקלוט את אור השמש."

דוגמה 3: עוזר אישי

הוראה: "תקבע פגישה ליום שלישי הבא בשעה 3 אחר הצהריים."

שאלה בכתיבה אנושית: "האם תוכל לקבוע פגישה ליום שלישי בשעה 3?"

תשובה בכתיבה אנושית: "בוודאי, קבעתי עבורך פגישה ליום שלישי הבא בשעה 3 אחר הצהריים. האם אתה צריך משהו נוסף?"

התוצאה היא מערכת אקולוגית מגוונת של מודלי שפה גדולים בגדלים שונים ועם התמחויות שונות. מודלים קטנים יותר בטווח של 1-7 מיליארד פרמטרים מספקים יכולות שפה כלליות טובות תוך שמירה על יעילות בהפעלה.

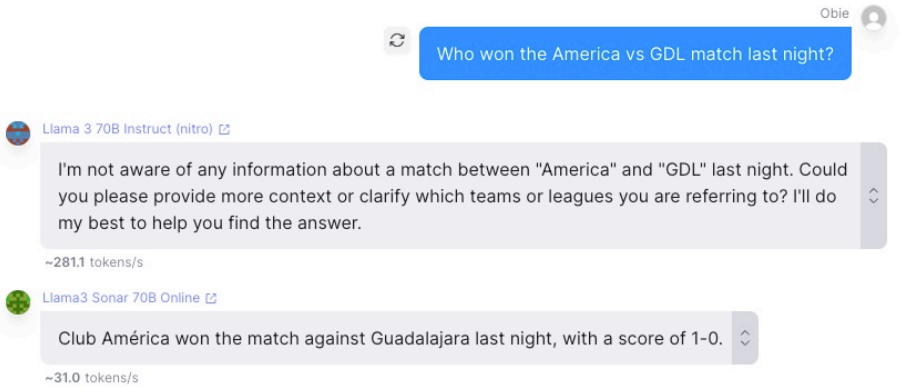
- 7B Mistral
- 8B 3 Llama
- 7B Gemma

מודלים בגודל בינוני של כ-30-70 מיליארד פרמטרים מציעים יכולות הסקה ומעקב אחר הוראות חזקות יותר.

- 70B 3 Llama
- 70B Qwen2
- 8x22B Mixtral

בעת בחירת מודל שפה גדול לשילוב באפליקציה, עליך לאזן בין יכולות המודל לבין גורמים מעשיים כמו עלות, זמן תגובה, אורך הקונטקסט וסינון תוכן. מודלים קטנים יותר שעברו כוונון הוראות הם לרוב הבחירה הטובה ביותר למשימות שפה פשוטות יותר, בעוד שהמודלים הגדולים ביותר עשויים להידרש להסקה או ניתוח מורכבים. נתוני האימון של המודל הם גם שיקול חשוב, שכן הם קובעים את תאריך חתך הידע של המודל.

מודלים מסוימים, כמו חלק מאלה של Perplexity מחוברים למקורות מידע בזמן אמת, כך שלמעשה אין להם תאריך חתך. כאשר שואלים אותם שאלות, הם מסוגלים להחליט באופן עצמאי לבצע חיפושים באינטרנט ולאחזר דפי אינטרנט שונים כדי ליצור תשובה.



איור 3.1. Llama3 עם ובלי גישה מקוונת

בסופו של דבר, אין מודל שפה גדול (LLM) אחד שמתאים לכל המטרות. הבנת ההבדלים בגודל המודל, בארכיטקטורה ובאימון היא מפתח לבחירת המודל המתאים לשימוש הנדרש. ניסוי במודלים שונים הוא הדרך המעשית היחידה לגלות אילו מודלים מספקים את הביצועים הטובים ביותר למשימה הנתונה.

טוקניזציה: פירוק טקסט לחלקים

לפני שמודל שפה גדול יכול לעבד טקסט, יש לפרק את הטקסט ליחידות קטנות יותר הנקראות טוקנים. טוקנים יכולים להיות מילים בודדות, חלקי מילים, ואפילו תווים בודדים. תהליך פירוק הטקסט לטוקנים נקרא טוקניזציה, וזהו שלב קריטי בהכנת הנתונים עבור מודל השפה.

איור 3.2. משפט זה מכיל 27 טוקנים

מודלי שפה גדולים שונים משתמשים באסטרטגיות טוקניזציה שונות, אשר יכולות להשפיע באופן משמעותי על ביצועי המודל ויכולותיו. כמה מהטוקנייזרים הנפוצים המשמשים במודלי שפה גדולים כוללים:

- **GPT (קידוד זוגות בתים):** טוקנייזרים של GPT משתמשים בטכניקה הנקראת קידוד זוגות בתים BPE)) כדי לפרק טקסט ליחידות תת-מילים. BPE מאחד באופן חוזר את הזוגות השכיחים ביותר של בתים במאגר הטקסט, ויוצר אוצר מילים של טוקנים תת-מיליים. זה מאפשר לטוקנייזר לטפל במילים נדירות וחדשות על ידי פירוקן לחלקי תת-מילים נפוצים יותר. טוקנייזרים של GPT משמשים במודלים כמו GPT-3 ו-GPT-4.

- **Llama (SentencePiece):** טוקנייזרים של Llama משתמשים בספריית SentencePiece, שהיא טוקנייזר ודה-טוקנייזר טקסט ללא פיקוח. SentencePiece מתייחס לטקסט הקלט כרצף של תווי יוניקוד ולומד אוצר מילים של תת-מילים על בסיס מאגר אימון. הוא יכול לטפל בכל שפה שניתן לקודד ביוניקוד, מה שהופך אותו למתאים במיוחד למודלים רב-לשוניים. טוקנייזרים של Llama משמשים במודלים כמו Llama ו-Alpaca של Meta.

- **SentencePiece (Unigram):** מטקני SentencePiece יכולים להשתמש גם באלגוריתם שונה הנקרא Unigram, המבוסס על טכניקת רגולריזציה של תתי-מילים. טקניזציית Unigram קובעת את אוצר תתי-המילים האופטימלי בהתבסס על מודל שפה מסוג יוניגרם, המקצה הסתברויות ליחידות תת-מילים בודדות. גישה זו יכולה לייצר תתי-מילים בעלות משמעות סמנטית רבה יותר בהשוואה ל-BPE. SentencePiece עם Unigram משמש מודלים כמו T5 של Google ו-BERT.

- **Gemini Google (טקניזציה רב-אופנית):** Gemini Google משתמש בשיטת טקניזציה המתוכננת לטפל בסוגי נתונים שונים, כולל טקסט, תמונות, שמע, וידאו וקוד. יכולת רב-אופנית זו מאפשרת ל-Gemini לעבד ולשלב צורות מידע שונות. באופן בולט, ל-Gemini Google Pro 1.5 יש חלון הקשר שיכול לטפל במיליוני טוקנים, הרבה יותר מאשר מודלים קודמים. חלון ההקשר הנרחב הזה מאפשר למודל לעבד הקשר גדול יותר, מה שעשוי להוביל לתשובות מדויקות יותר. עם זאת, חשוב לציין ששיטת הטקניזציה של Gemini קרובה הרבה יותר לטוקן אחד לכל תו מאשר מודלים אחרים. משמעות הדבר היא שהעלות בפועל של שימוש במודלים של Gemini עשויה להיות גבוהה משמעותית מהמצופה אם אתם רגילים להשתמש במודלים כמו GPT, מכיוון שהתמחור של Google מבוסס על תווים ולא על טוקנים.

בחירת המטקן משפיעה על מספר היבטים של מודל LLM, כולל:

- **גודל אוצר המילים:** המטקן קובע את גודל אוצר המילים של המודל, שהוא אוסף הטוקנים הייחודיים שהוא מזהה. אוצר מילים גדול יותר ומפורט יותר יכול לעזור למודל להתמודד עם מגוון רחב יותר של מילים וביטויים ואפילו להפוך לרב-אופני (מסוגל להבין ולייצר יותר מאשר רק טקסט), אך הוא גם מגדיל את דרישות הזיכרון והמורכבות החישובית של המודל.
- **טיפול במילים נדירות ולא מוכרות:** מטקנים המשתמשים ביחידות תת-מילים, כמו BPE ו-SentencePiece, יכולים לפרק מילים נדירות ולא מוכרות לחלקי תת-מילים נפוצים יותר. זה מאפשר למודל לנחש ניחושים מושכלים לגבי המשמעות של מילים שלא ראה קודם, בהתבסס על תתי-המילים שהן מכילות.
- **תמיכה רב-לשונית:** מטקנים כמו SentencePiece, שיכולים לטפל בכל שפה הניתנת לקידוד ביוניקוד, מתאימים היטב למודלים רב-לשוניים שצריכים לעבד טקסט במספר שפות.

בבחירת מודל שפה גדול (LLM) ליישום מסוים, חשוב לשקול את מפענח הטוקנים בו הוא משתמש ועד כמה הוא מתאים לצרכי עיבוד השפה הספציפיים של המשימה הנדונה. למפענח הטוקנים יכולה להיות השפעה משמעותית על יכולת המודל להתמודד עם מונחים ייחודיים לתחום, מילים נדירות וטקסט רב-לשוני.

גודל הקונטקסט: כמה מידע יכול מודל שפה לנצל במהלך הסקה?

כאשר דנים במודלי שפה, גודל הקונטקסט מתייחס לכמות הטקסט שמודל יכול לקחת בחשבון בעת עיבוד או יצירת תשובותיו. זוהי למעשה מדידה של כמות המידע שהמודל יכול "לזכור" ולהשתמש בו כדי להשפיע על הפלט שלו (מבוטא בטוקנים). לגודל הקונטקסט של מודל שפה יכולה להיות השפעה משמעותית על יכולותיו וסוגי המשימות שהוא יכול לבצע ביעילות.

מהו גודל הקונטקסט?

במונחים טכניים, גודל הקונטקסט נקבע על פי מספר הטוקנים (מילים או חלקי מילים) שמודל שפה יכול לעבד ברצף קלט יחיד. לעיתים קרובות מתייחסים לכך כ"טווח הקשב" או "חלון

ההקשר" של המודל. ככל שגודל הקונטקסט גדול יותר, כך המודל יכול לקחת בחשבון יותר טקסט בבת אחת בעת יצירת תשובה או ביצוע משימה.

למודלי שפה שונים יש גדלי קונטקסט משתנים, הנעים ממאות בודדות של טוקנים ועד למיליוני טוקנים. לשם השוואה, פסקה טיפוסית של טקסט עשויה להכיל כ-100-150 טוקנים, בעוד שספר שלם עשוי להכיל עשרות או מאות אלפי טוקנים.

> קיימת אפילו עבודה על שיטות יעילות להרחבת מודלי שפה גדולים (LLMs) מבוססי טרנספורמר לקלט באורך אינסופי עם זיכרון וחישוב מוגבלים.

מדוע גודל הקונטקסט חשוב?

לגודל הקונטקסט של מודל שפה יש השפעה משמעותית על יכולתו להבין וליצור טקסט קוהרנטי ורלוונטי להקשר. הנה כמה סיבות מרכזיות לחשיבותו של גודל הקונטקסט:

1. **הבנת תוכן ארוך:** מודלים עם גודל קונטקסט גדול יותר יכולים להבין ולנתח טוב יותר טקסטים ארוכים, כמו מאמרים, דוחות, ואפילו ספרים שלמים. זה חיוני למשימות כמו תמצות מסמכים, מענה על שאלות, וניתוח תוכן.
2. **שמירה על לכידות:** חלון הקשר גדול יותר מאפשר למודל לשמור על לכידות ועקביות לאורך קטעי פלט ארוכים יותר. זה חשוב למשימות כמו יצירת סיפורים, מערכות דיאלוג, ויצירת תוכן, שבהן שמירה על נרטיב או נושא עקבי היא חיונית. זה גם קריטי לחלוטין כאשר משתמשים במודלי שפה גדולים ליצירה או להמרה של מידע מובנה.
3. **לכידת תלויות ארוכות טווח:** חלק ממשימות השפה דורשות הבנה של קשרים בין מילים או ביטויים המרוחקים זה מזה בטקסט. מודלים עם גודל הקשר גדול יותר מצוידים טוב יותר ללכידת תלויות ארוכות טווח אלה, שיכולות להיות חשובות למשימות כמו ניתוח רגשות, תרגום, והבנת שפה.
4. **טיפול בהוראות מורכבות:** ביישומים בהם נעשה שימוש במודלי שפה כדי לעקוב אחר הוראות מורכבות מרובות שלבים, גודל הקשר גדול יותר מאפשר למודל להתחשב במערך ההוראות המלא בעת יצירת תגובה, במקום רק במילים האחרונות.

דוגמאות למודלי שפה עם גדלי הקשר שונים

הנה מספר דוגמאות של מודלי שפה עם גדלי הקשר שונים:

- OpenAI GPT-3.5 Turbo: 4,095 טוקנים
- Mistral 7B Instruct: 32,768 טוקנים
- Anthropic Claude v1: 100,000 טוקנים
- OpenAI GPT-4 Turbo: 128,000 טוקנים
- Anthropic Claude v2: 200,000 טוקנים
- Google Gemini Pro 1.5: 2.8M טוקנים

כפי שניתן לראות, קיים טווח רחב של גדלי הקשר בין המודלים האלה, החל מכ-4,000 טוקנים עבור מודל OpenAI GPT-3.5 Turbo ועד 200,000 טוקנים עבור מודל Anthropic Claude v2. חלק מהמודלים, כמו Google's PaLM 2 ו-OpenAI's GPT-4, מציעים גרסאות שונות עם גדלי הקשר גדולים יותר (למשל, גרסאות "32k"), שיכולות לטפל ברצפי קלט ארוכים אף יותר. ונכון לרגע זה (אפריל 2024) Google Gemini Pro 1.5 מתהדר בכמעט 3 מיליון טוקנים!

חשוב לציין שגודל ההקשר יכול להשתנות בהתאם למימוש ולגרסה הספציפיים של מודל מסוים. לדוגמה, למודל המקורי OpenAI GPT-4 יש גודל הקשר של 8,191 טוקנים, בעוד שלגרסאות המאוחרות יותר של GPT-4 כמו Turbo ו-4o יש גודל הקשר גדול בהרבה של 128,000 טוקנים.

> סם אלטמן השווה את מגבלות הקונטקסט הנוכחיות לקילובייטים של זיכרון העבודה שעימם התמודדו מתכנתי המחשבים האישיים בשנות ה-80, ואמר שבעתיד הקרוב נוכל להכניס את "כל המידע האישי שלך" לתוך הקונטקסט של מודל שפה גדול.

בחירת גודל הקונטקסט המתאים

בעת בחירת מודל שפה ליישום מסוים, חשוב לשקול את דרישות גודל הקונטקסט של המשימה הנדונה. עבור משימות הכוללות קטעי טקסט קצרים ומבודדים, כמו ניתוח רגשות או מענה

פשוט על שאלות, ייתכן שגודל קונטקסט קטן יותר יספיק. עם זאת, עבור משימות הדורשות הבנה ויצירה של טקסטים ארוכים ומורכבים יותר, סביר להניח שיידרש גודל קונטקסט גדול יותר.

ראוי לציין שגדלי קונטקסט גדולים יותר מלווים לעתים קרובות בעלויות חישוביות מוגדלות וזמני עיבוד איטיים יותר, מכיוון שהמודל צריך להתחשב ביותר מידע בעת יצירת תגובה. לפיכך, עליך למצוא איזון בין גודל הקונטקסט לביצועים בעת בחירת מודל שפה ליישום שלך.

למה לא פשוט לבחור את המודל עם גודל הקונטקסט הגדול ביותר ולדחוס לתוכו כמה שיותר מידע? ובכן, מלבד גורמי הביצועים, השיקול העיקרי האחר הוא העלות. במרץ 2024, מחזור בקשה-תגובה בגודל באמצעות Google Gemini 5.1 Pro עם קונטקסט מלא יעלה לך כמעט 8 דולר .(USD) אם יש לך מקרה שימוש שמצדיק הוצאה כזו, כל הכבוד! אבל עבור רוב היישומים, זה פשוט יקר מדי בסדרי גודל.

מציאת מחטים בערימות השחת

המושג של מציאת מחט בערימת שחת היה מאז ומתמיד משל לאתגרי האחזור במאגרי נתונים גדולים. בתחום מודלי השפה הגדולים, אנו מתאימים מעט את האנלוגיה הזו. דמיינו שאנחנו לא מחפשים רק עובדה בודדת הקבורה בתוך טקסט עצום (כמו אסופה מלאה של מאמרי פול גרהאם), אלא עובדות מרובות המפוזרות לאורכו. תרחיש זה דומה יותר למציאת כמה מחטים בשדה נרחב, ולא רק בערימת שחת אחת. והנה העניין: לא רק שעלינו לאתר את המחטים הללו, אלא גם עלינו לשזור אותן לכדי חוט קוהרנטי.

כאשר מודלי שפה גדולים מתמודדים עם המשימה של אחזור וחשיבה על עובדות מרובות המוטמעות בהקשרים ארוכים, הם ניצבים בפני אתגר כפול. ראשית, ישנה הסוגיה הפשוטה של דיוק האחזור - הוא באופן טבעי יורד ככל שמספר העובדות גדל. זה צפוי; אחרי הכל, מעקב אחר פרטים מרובים לאורך טקסט נרחב מכביד אפילו על המודלים המתוחכמים ביותר.

שנית, ואולי באופן קריטי יותר, ישנו האתגר של הסקת מסקנות מעובדות אלה. זה דבר

אחד לבחור עובדות; דבר אחר לגמרי הוא לסנתז אותן לכדי נרטיב או תשובה קוהרנטיים. כאן מגיע המבחן האמיתי. הביצועים של מודלי שפה גדולים (LLMs) במשימות הסקה נוטים להידרדר יותר מאשר במשימות אחזור פשוטות. הידרדרות זו אינה רק עניין של נפח; מדובר בריקוד המורכב של הקשר, רלוונטיות והסקה.

מדוע זה קורה? ובכן, חשבו על הדינמיקה של זיכרון וקשב בקוגניציה האנושית, המשתקפת במידה מסוימת במודלי שפה גדולים. בעת עיבוד כמויות גדולות של מידע, מודלי שפה גדולים, בדומה לבני אדם, עלולים לאבד את העקבות של פרטים מוקדמים בעודם קולטים חדשים. הדבר נכון במיוחד במודלים שלא תוכננו במפורש לתעדף או לחזור באופן אוטומטי לקטעי טקסט מוקדמים.

יתר על כן, היכולת של מודל שפה גדול לארוג את העובדות שאוחזרו לכדי תגובה קוהרנטית דומה לבניית נרטיב. זה דורש לא רק אחזור מידע אלא הבנה עמוקה ומיקום הקשרי, שנותר אתגר קשה עבור הבינה המלאכותית הנוכחית.

אז מה זה אומר לנו כמפתחים ומשלבים של טכנולוגיות אלה? עלינו להיות מודעים היטב למגבלות אלה בעת תכנון מערכות המסתמכות על מודלי שפה גדולים לטיפול במשימות מורכבות וארוכות. ההבנה שהביצועים עשויים להידרדר בתנאים מסוימים עוזרת לנו להציב ציפיות ריאליסטיות ולתכנן מנגנוני גיבוי או אסטרטגיות משלימות טובים יותר.

אופנויות: מעבר לטקסט

בעוד שרוב מודלי השפה כיום מתמקדים בעיבוד ויצירת טקסט, ישנה מגמה גוברת לעבר מודלים מרובי-אופנויות שיכולים באופן טבעי לקלוט ולהפיק סוגים שונים של מידע, כגון תמונות, אודיו ווידאו. מודלים מרובי-אופנויות אלה פותחים אפשרויות חדשות ליישומים מבוססי בינה מלאכותית שיכולים להבין וליצור תוכן במגוון אופנויות.

מהן אופנויות?

בהקשר של מודלי שפה, אופנויות מתייחסות לסוגים השונים של מידע שמודל יכול לעבד וליצור. האופנות הנפוצה ביותר היא טקסט, הכולל שפה כתובה בצורות שונות כמו ספרים, מאמרים, אתרי אינטרנט ופוסטים ברשתות חברתיות. עם זאת, ישנן מספר אופנויות אחרות שמשולבות יותר ויותר במודלי שפה:

- **תמונות:** מידע חזותי כגון תצלומים, איורים ותרשימים.
- **אודיו:** מידע קולי כגון דיבור, מוזיקה וקולות סביבתיים.
- **וידאו:** מידע חזותי נע, לרוב מלווה באודיו, כגון קטעי וידאו וסרטים.

כל אופנות מציגה אתגרים והזדמנויות ייחודיים עבור מודלים שפתיים. למשל, תמונות דורשות מהמודל להבין מושגים ויחסים חזותיים, בעוד שאודיו דורש מהמודל לעבד ולייצר דיבור וצלילים אחרים.

מודלים שפתיים מרובי-אופנויות

מודלים שפתיים מרובי-אופנויות מתוכננים לטפל במספר אופנויות בתוך מודל יחיד. מודלים אלה כוללים בדרך כלל רכיבים או שכבות מתמחות שיכולות גם להבין קלט וגם לייצר פלט בסוגי אופנויות שונים. כמה דוגמאות בולטות של מודלים שפתיים מרובי-אופנויות כוללות:

- **GPT-4o של OpenAI:** GPT-4o הוא מודל שפתי גדול המבין ומעבד באופן טבעי אודיו דיבורי בנוסף לטקסט. יכולת זו מאפשרת ל-GPT-4o לבצע משימות כמו תמלול שפה מדוברת, יצירת טקסט מקלט אודיו, ומתן תשובות המבוססות על שאילתות מדוברות.
- **GPT-4 של OpenAI עם קלט חזותי:** GPT-4 הוא מודל שפתי גדול המסוגל לעבד הן טקסט והן תמונות. כאשר ניתנת תמונה כקלט, GPT-4 יכול לנתח את תוכן התמונה ולייצר טקסט המתאר או מגיב למידע החזותי.
- **Gemini של Google:** Gemini הוא מודל מרובה-אופנויות המסוגל לטפל בטקסט, תמונות ווידאו. הוא משתמש בארכיטקטורה מאוחדת המאפשרת הבנה ויצירה חוצות-אופנויות, ומאפשר משימות כמו תיאור תמונות, סיכום וידאו ומענה לשאלות חזותיות.

- **DALL-E ו-Stable Diffusion:** למרות שאינם מודלים שפתיים במובן המסורתי, מודלים אלה מדגימים את עוצמת הבינה המלאכותית מרובת-האופנויות על ידי יצירת תמונות מתיאורים טקסטואליים. הם מציגים את הפוטנציאל של מודלים שיכולים לתרגם בין אופנויות שונות.

יתרונות ויישומים של מודלים מרובי-אופנויות

מודלים שפתיים מרובי-אופנויות מציעים מספר יתרונות ומאפשרים מגוון רחב של יישומים, כולל:

- **הבנה משופרת:** על ידי עיבוד מידע ממספר אופנויות, מודלים אלה יכולים להשיג הבנה מקיפה יותר של העולם, בדומה לאופן שבו בני אדם לומדים ממגוון קלטים חושיים.
- **יצירה חוצת-אופנויות:** מודלים מרובי-אופנויות יכולים לייצר תוכן באופנות אחת על בסיס קלט מאופנות אחרת, כמו יצירת תמונה מתיאור טקסטואלי או יצירת סיכום וידאו ממאמר כתוב.
- **נגישות:** מודלים מרובי-אופנויות יכולים להפוך מידע לנגיש יותר על ידי תרגום בין אופנויות, כמו יצירת תיאורי טקסט של תמונות עבור משתמשים עם לקות ראייה או יצירת גרסאות אודיו של תוכן כתוב.
- **יישומים יצירתיים:** ניתן להשתמש במודלים מרובי-מודליות למשימות יצירתיות כמו יצירת אמנות, מוזיקה או סרטונים על בסיס הנחיות טקסטואליות, דבר הפותח אפשרויות חדשות עבור אמנים ויוצרי תוכן.

ככל שמודלים שפתיים מרובי-מודליות ממשיכים להתקדם, הם צפויים למלא תפקיד חשוב יותר ויותר בפיתוח יישומים מבוססי בינה מלאכותית המסוגלים להבין וליצור תוכן במספר מודליות. הדבר יאפשר אינטראקציות טבעיות ואינטואיטיביות יותר בין בני אדם ומערכות בינה מלאכותית, וכן יפתח אפשרויות חדשות לביטוי יצירתי והפצת ידע.

מערכות אקולוגיות של ספקים

כאשר מדובר בשילוב מודלים שפתיים גדולים (LLMs) ביישומים, עומד לרשותך מגוון הולך וגדל של אפשרויות לבחירה. כל ספק LLM מרכזי, כגון OpenAI, Anthropic, Google, ו-Cohere, מציע מערכת אקולוגית משלו של מודלים, ממשקי API וכלים. בחירת הספק המתאים כרוכה בשקלול גורמים שונים, כולל תמחור, ביצועים, סינון תוכן, פרטיות מידע ואפשרויות התאמה אישית.

OpenAI

OpenAI היא אחת מספקיות ה-LLM המוכרות ביותר, כאשר סדרת ה-GPT שלה (GPT-3, GPT-4) נמצאת בשימוש נרחב במגוון יישומים. OpenAI מציעה ממשק API ידידותי למשתמש המאפשר לשלב בקלות את המודלים שלהם ביישומים. הם מספקים מגוון מודלים עם יכולות ומחירים שונים, החל מהמודל הבסיסי Ada ועד למודל החזק Davinci.

המערכת האקולוגית של OpenAI כוללת גם כלים כמו Playground, OpenAI המאפשר להתנסות בהנחיות ולבצע כוונון עדין של מודלים עבור מקרי שימוש ספציפיים. הם מציעים אפשרויות סינון תוכן כדי לסייע במניעת יצירת תוכן לא הולם או מזיק.

כאשר משתמשים במודלים של OpenAI באופן ישיר, אני מסתמך על ספריית ruby-openai של Rudall Alex.

Anthropic

Anthropic היא שחקנית מרכזית נוספת בתחום ה-LLM, כאשר מודלי Claude שלה צוברים פופולריות בזכות ביצועים חזקים ושיקולים אתיים. Anthropic מתמקדת בפיתוח מערכות בינה מלאכותית בטוחות ואחראיות, עם דגש חזק על סינון תוכן והימנעות מפלט מזיק.

המערכת האקולוגית של Anthropic כוללת את ממשק ה-API של Claude, המאפשר לשלב את המודל ביישומים שלהם, וכן כלים להנדסת הנחיות וכוונון עדין. הם מציעים גם את מודל Instant, Claude המשלב יכולות חיפוש באינטרנט לקבלת תשובות עדכניות ומדויקות יותר.

בעת שימוש במודלים של Anthropic באופן ישיר, אני מסתמך על ספריית anthrophic של Alex Rudall.

Google

Google פיתחה מספר מודלי שפה גדולים (LLMs) חזקים, כולל Gemini, BERT, T5, ו-PaLM. מודלים אלה ידועים בביצועים החזקים שלהם במגוון רחב של משימות עיבוד שפה טבעית. המערכת האקולוגית של Google כוללת את ספריות TensorFlow ו-Keras, המספקות כלים ומסגרות עבודה לבניה ואימון של מודלים ללמידת מכונה.

Google מציעה גם פלטפורמת בינה מלאכותית בענן, המאפשרת לפרוס ולהרחיב בקלות את המודלים שלהם בענן. הם מספקים מגוון של מודלים מאומנים מראש ו-APIs למשימות כמו ניתוח רגשות, זיהוי ישויות ותרגום.

Meta

Meta, שנודעה בעבר כ-Facebook, משקיעה רבות בפיתוח מודלי שפה גדולים, כפי שמודגש בשחרור מודלים כמו LLaMA ו-OPT. מודלים אלה בולטים בביצועיהם החזקים במגוון משימות שפה וזמינים בעיקר דרך ערוצי קוד פתוח, תומכים במחויבות של Meta למחקר ושיתוף פעולה קהילתי.

המערכת האקולוגית של Meta בנויה בעיקר סביב PyTorch, ספריית למידת מכונה בקוד פתוח המועדפת בזכות יכולות החישוב הדינמיות והגמישות שלה, המאפשרת מחקר ופיתוח חדשניים בתחום הבינה המלאכותית.

בנוסף להצעות הטכניות שלהם, Meta שמה דגש חזק על פיתוח בינה מלאכותית אתית. הם מיישמים סינון תוכן מקיף ומתמקדים בהפחתת הטיות, בהתאמה למטרותיהם הרחבות של בטיחות ואחריות ביישומי בינה מלאכותית.

Cohere

Cohere היא שחקנית חדשה יחסית בתחום ה-LLM, המתמקדת בהנגשת מודלי שפה גדולים ובהפיכתם לקלים יותר לשימוש מהמתחרים. המערכת האקולוגית שלהם כוללת את Cohere API, המספק גישה למגוון מודלים מאומנים מראש למשימות כמו יצירת טקסט, סיווג וסיכום.

Cohere מציעה גם כלים להנדסת פרומפטים, כוונון עדין וסינון תוכן. הם מדגישים פרטיות ואבטחת מידע, עם תכונות כמו אחסון מוצפן ובקרות גישה.

Ollama

Ollama היא פלטפורמה מאוחסנת עצמאית המאפשרת למשתמשים לנהל ולפרוס מגוון מודלי שפה גדולים (LLMs) באופן מקומי על המכונות שלהם, ומעניקה להם שליטה מלאה על מודלי הבינה המלאכותית שלהם ללא תלות בשירותי ענן חיצוניים. הגדרה זו אידיאלית עבור אלה

שמעדיפים פרטיות נתונים ומעוניינים לנהל את פעולות הבינה המלאכותית שלהם באופן פנימי.

הפלטפורמה תומכת במגוון מודלים, כולל גרסאות של Gemma Phi, Llama, ו-Mistral, אשר נבדלים בגודלם ובדרישות החישוביות שלהם. Ollama מאפשר להוריד ולהריץ מודלים אלה ישירות משורת הפקודה באמצעות פקודות פשוטות כמו ollama run model_<name>, והוא מתוכנן לעבוד במערכות הפעלה שונות כולל macOS Linux ו-Windows.

עבור מפתחים המעוניינים לשלב מודלים בקוד פתוח באפליקציות שלהם מבלי להשתמש ב-API מרוחק, Ollama מציע ממשק שורת פקודה לניהול מחזור החיים של המודלים, בדומה לכלי ניהול מכולות. הוא גם תומך בתצורות והנחיות מותאמות אישית, המאפשרות רמה גבוהה של התאמה אישית כדי להתאים את המודלים לצרכים או לשימושים ספציפיים.

Ollama מתאים במיוחד למשתמשים בעלי ידע טכני ולמפתחים בזכות ממשק שורת הפקודה שלו והגמישות שהוא מציע בניהול ופריסת מודלים של בינה מלאכותית. זה הופך אותו לכלי רב-עוצמה לעסקים ויחידים הזקוקים ליכולות בינה מלאכותית חזקות מבלי להתפשר על אבטחה ושליטה.

פלטפורמות מרובות-מודלים

בנוסף, ישנם ספקים המארחים מגוון רחב של מודלים בקוד פתוח, כגון Together.ai ו-Groq. פלטפורמות אלה מציעות גמישות והתאמה אישית, המאפשרות להריץ ובמקרים מסוימים אפילו לבצע כוונון עדין של מודלים בקוד פתוח בהתאם לצרכים הספציפיים שלך. לדוגמה, Together.ai מספק גישה למגוון של מודלי LLM בקוד פתוח, המאפשרים למשתמשים להתנסות במודלים ותצורות שונות. Groq מתמקד באספקת השלמת טקסט בביצועים גבוהים במיוחד שבזמן כתיבת ספר זה נראית כמעט קסומה

בחירת ספק LLM

בעת בחירת ספק LLM, עליך לשקול גורמים כגון:

- **תמחור:** ספקים שונים מציעים מודלים שונים של תמחור, החל מתשלום לפי שימוש ועד תוכניות מבוססות מנוי. חשוב לשקול את השימוש הצפוי והתקציב בעת בחירת ספק.

- **ביצועים:** ביצועי ה-LLM יכולים להשתנות משמעותית בין ספקים, לכן חשוב לבצע בדיקות השוואה ולבחון מודלים על מקרי שימוש ספציפיים לפני קבלת החלטה.
- **סינון תוכן:** בהתאם לאפליקציה, סינון תוכן עשוי להיות שיקול קריטי. חלק מהספקים מציעים אפשרויות סינון תוכן חזקות יותר מאחרים.
- **פרטיות מידע:** אם האפליקציה מטפלת במידע רגיש של משתמשים, חשוב לבחור ספק עם נהלי פרטיות ואבטחת מידע חזקים.
- **התאמה אישית:** חלק מהספקים מציעים יותר גמישות מבחינת כוונון עדין והתאמה אישית של מודלים למקרי שימוש ספציפיים.

בסופו של דבר, הבחירה של ספק מודל השפה הגדול תלויה בדרישות ובאילוצים הספציפיים של היישום. באמצעות הערכה זהירה של האפשרויות והתחשבות בגורמים כמו תמחור, ביצועים ופרטיות נתונים, תוכלו לבחור את הספק שעונה בצורה הטובה ביותר על צרכיכם.

חשוב לציין גם שנוף מודלי השפה הגדולים מתפתח ללא הרף, עם ספקים ומודלים חדשים שמופיעים באופן קבוע. כדאי להישאר מעודכנים בהתפתחויות האחרונות ולהיות פתוחים לחקור אפשרויות חדשות כשהן הופכות זמינות.

OpenRouter

לאורך ספר זה אסתמך באופן בלעדי על OpenRouter כספק ה-API המועדף עליי. הסיבה פשוטה: זוהי חנות אחת שמרכזת את כל המודלים המסחריים והקוד הפתוח הפופולריים ביותר. אם אתם משתוקקים להתחיל להתעסק עם קידוד בינה מלאכותית, אחד המקומות הטובים ביותר להתחיל בהם הוא עם ספריית הרובי של OpenRouter שפיתחתי.

חשיבה על ביצועים

כאשר משלבים מודלי שפה ביישומים, ביצועים הם שיקול קריטי. ניתן למדוד את ביצועי מודל השפה במונחים של שהייה (הזמן שלוקח לייצר תגובה) ותפוקה (מספר הבקשות שהוא יכול לטפל בהן ביחידת זמן).

זמן עד לאסימון הראשון (TTFT) הוא מדד ביצועים חיוני נוסף, הרלוונטי במיוחד לצ'טבוטים ויישומים הדורשים תגובות אינטראקטיביות בזמן אמת. TTFT מודד את השהייה מהרגע

שבקשת המשתמש מתקבלת ועד לרגע שהמילה (או האסימון) הראשונה של התגובה מיוצרת. מדד זה הוא קריטי לשמירה על חוויית משתמש חלקה ומעורבת, שכן תגובות מושהות עלולות להוביל לתסכול המשתמש וניתוק.

למדדי ביצועים אלה יכולה להיות השפעה משמעותית על חוויית המשתמש והיכולת להרחיב את היישום.

מספר גורמים יכולים להשפיע על ביצועי מודל השפה, כולל:

מספר פרמטרים: מודלים גדולים יותר עם יותר פרמטרים בדרך כלל דורשים יותר משאבי מחשוב ויכולים להיות בעלי שהייה גבוהה יותר ותפוקה נמוכה יותר בהשוואה למודלים קטנים יותר.

חומרה: ביצועי מודל השפה יכולים להשתנות משמעותית בהתבסס על החומרה שעליה הוא רץ. ספקי ענן מציעים שרתי מעבד גרפי ומעבד טנסור המותאמים לעומסי למידת מכונה, אשר יכולים להאיץ מאוד את היסק המודל.

אחד הדברים הנחמדים ב-OpenRouter הוא שעבור רבים מהמודלים שהוא מציע, יש לך בחירה בין ספקי ענן עם מגוון פרופילי ביצועים ועלויות.

קוונטיזציה: ניתן להשתמש בטכניקות קוונטיזציה כדי להפחית את טביעת הרגל בזיכרון ואת דרישות החישוב של מודל על ידי ייצוג משקולות והפעלות בטיפוסי נתונים בדיוק נמוך יותר. זה יכול לשפר את הביצועים מבלי לפגוע משמעותית באיכות. כמפתח יישומים, כנראה שלא תהיה מעורב באימון המודלים שלך ברמות קוונטיזציה שונות, אבל כדאי להכיר לפחות את המונחים.

אצווה: עיבוד מספר בקשות בו-זמנית באצוות יכול לשפר את התפוקה על ידי פיזור התקורה של טעינת המודל והעברת הנתונים.

מטמון: שמירת התוצאות של הנחיות או רצפי קלט שנמצאים בשימוש תכוף במטמון יכולה להפחית את מספר בקשות ההיסק ולשפר את הביצועים הכוללים.

בעת בחירת מודל שפה ליישום בייצור, חשוב לבדוק את ביצועיו על עומסי עבודה ותצורות חומרה מייצגים. זה יכול לעזור לזהות צווארי בקבוק פוטנציאליים ולהבטיח שהמודל יכול לעמוד ביעדי הביצועים הנדרשים.

כדאי גם לשקול את האיזון בין ביצועי המודל לבין גורמים אחרים כמו עלות, גמישות וקלות השילוב. למשל, שימוש במודל קטן וזול יותר עם השהייה נמוכה יותר עשוי להיות עדיף עבור יישומים הדורשים תגובות בזמן אמת, בעוד שמודל גדול וחזק יותר עשוי להתאים יותר לעיבוד אצווה או משימות חשיבה מורכבות.

התנסות עם מודלי LLM שונים

בחירת LLM היא לעתים רחוקות החלטה קבועה. מכיוון שמודלים חדשים ומשופרים משוחררים באופן קבוע, כדאי לבנות יישומים בצורה מודולרית המאפשרת להחליף מודלי שפה שונים לאורך זמן. ניתן לעתים קרובות לעשות שימוש חוזר בהנחיות ובמערכי נתונים בין מודלים עם שינויים מינימליים. זה מאפשר לך לנצל את ההתקדמות האחרונה במידול שפה מבלי לעצב מחדש לחלוטין את היישומים שלך.

היכולת להחליף בקלות בין מגוון רחב של אפשרויות מודלים היא עוד סיבה שאני אוהב את OpenRouter.

בעת שדרוג למודל שפה חדש, חשוב לבדוק ולאמת ביסודיות את ביצועיו ואיכות הפלט שלו כדי להבטיח שהוא עומד בדרישות היישום. זה עשוי לכלול אימון מחדש או כוונון עדין של המודל על נתונים ספציפיים לתחום, וכן עדכון של רכיבי המורד שתלויים בפלטי המודל.

על ידי תכנון יישומים תוך התחשבות בביצועים ומודולריות, תוכל ליצור מערכות הניתנות להרחבה, יעילות ועמידות לעתיד שיכולות להסתגל לנוף המתפתח במהירות של טכנולוגיית מידול השפה.

מערכות בינה מלאכותית מורכבות

לפני שנסיים את ההקדמה, חשוב לציין כי לפני 2023 והתפוצצות העניין בבינה מלאכותית יוצרת שהתעוררה בעקבות ChatGPT, גישות מסורתיות לבינה מלאכותית התבססו בדרך כלל על שילוב של מודלים בודדים וסגורים. לעומת זאת, מערכות בינה מלאכותית מורכבות מנצלות צינורות מורכבים של רכיבים מקושרים העובדים יחד להשגת התנהגות אינטליגנטית.

בליבן, מערכות בינה מלאכותית מורכבות מכילות מודולים מרובים, כשכל אחד מתוכנן לבצע משימות או פונקציות ספציפיות. מודולים אלה יכולים לכלול מחוללים, מאחזרים, מדרגים, ממיינים ורכיבים מתמחים נוספים. על ידי פירוק המערכת הכוללת ליחידות קטנות וממוקדות, מפתחים יכולים ליצור ארכיטקטורות בינה מלאכותית גמישות, מדרגות וברות-תחזוקה יותר.

אחד היתרונות המרכזיים של מערכות בינה מלאכותית מורכבות הוא יכולתן לשלב בין החוזקות של טכניקות ומודלים שונים של בינה מלאכותית. לדוגמה, מערכת עשויה להשתמש במודל שפה גדול (LLM) להבנה ויצירת שפה טבעית, תוך שימוש במודל נפרד לאחזור מידע או קבלת החלטות מבוססת-חוקים. גישה מודולרית זו מאפשרת לך לבחור את הכלים והטכניקות הטובים ביותר לכל משימה ספציפית, במקום להסתמך על פתרון אחיד לכל המצבים.

עם זאת, בניית מערכות בינה מלאכותית מורכבות מציבה גם אתגרים ייחודיים. במיוחד, הבטחת הקוהרנטיות והעקביות הכוללת של התנהגות המערכת דורשת מנגנוני בדיקה, ניטור וממשל חזקים.

הופעתם של מודלי שפה גדולים חזקים כמו GPT-4 מאפשרת לנו לערוך ניסויים במערכות בינה מלאכותית מורכבות בקלות רבה יותר מאי פעם, מכיוון שמודלים מתקדמים אלה מסוגלים לטפל בתפקידים מרובים בתוך מערכת מורכבת, כגון סיווג, דירוג ויצירה, בנוסף ליכולות הבנת השפה הטבעית שלהם. תכונת רב-גוניות זו מאפשרת למפתחים ליצור אבות-טיפוס ולשפר במהירות ארכיטקטורות בינה מלאכותית מורכבות, ופותחת אפשרויות חדשות לפיתוח יישומים אינטליגנטיים.

דפוסי פריסה למערכות בינה מלאכותית מורכבות

ניתן לפרוס מערכות בינה מלאכותית מורכבות באמצעות דפוסים שונים, כשכל אחד מתוכנן לענות על דרישות ומקרי שימוש ספציפיים. הבה נחקור ארבעה דפוסי פריסה נפוצים: שאלות ותשובות, פותרי בעיות מרובי-סוכנים, בינה מלאכותית שיחתית, ועוזרים אוטומטיים.

שאלות ותשובות

מערכות שאלות ותשובות (Q&A) מתמקדות באספקת אחזור מידע המועשר ביכולות ההבנה של מודלי בינה מלאכותית כדי לתפקד כיותר ממנוע חיפוש פשוט. על ידי שילוב מודלי שפה חזקים עם מקורות ידע חיצוניים באמצעות יצירה מועשרת באחזור (RAG), מערכות שאלות ותשובות נמנעות מהזיות ומספקות תשובות מדויקות ורלוונטיות להקשר לשאילתות המשתמש.

הרכיבים העיקריים של מערכת שאלות ותשובות מבוססת LLM כוללים:

- **הבנה ועיצוב מחדש של שאילתות:** ניתוח שאילתות המשתמש ועיצובן מחדש כדי להתאים טוב יותר למקורות הידע הבסיסיים.
- **אחזור מידע:** אחזור מידע רלוונטי ממקורות מידע מובנים או בלתי מובנים על בסיס השאילתה המעוצבת מחדש.
- **יצירת תשובות:** יצירת תשובות קוהרנטיות ואינפורמטיביות על ידי שילוב המידע שאוחזר עם יכולות היצירה של מודל השפה.

תת-מערכות RAG חשובות במיוחד בתחומי שאלות ותשובות שבהם מתן מידע מדויק ועדכני הוא קריטי, כגון תמיכת לקוחות, ניהול ידע, או יישומים חינוכיים

פותרי בעיות מרובי-סוכנים/אגנטיים

מערכות מרובות-סוכנים, הידועות גם כאגנטיות, מורכבות ממספר סוכנים אוטונומיים העובדים יחד לפתרון בעיות מורכבות. לכל סוכן יש תפקיד ספציפי, מערך כישורים וגישה לכלים או מקורות מידע רלוונטיים. באמצעות שיתוף פעולה וחילופי מידע, סוכנים אלה יכולים להתמודד עם משימות שהיו קשות או בלתי אפשריות לביצוע עבור סוכן יחיד.

העקרונות המרכזיים של פותרי בעיות מרובי-סוכנים כוללים:

- **התמחות:** כל סוכן מתמקד בהיבט ספציפי של הבעיה, תוך ניצול היכולות והידע הייחודיים שלו.
- **שיתוף פעולה:** הסוכנים מתקשרים ומתאמים את פעולותיהם להשגת מטרה משותפת, לרוב באמצעות העברת הודעות או זיכרון משותף.

- **יכולת הסתגלות:** המערכת יכולה להסתגל לתנאים או דרישות משתנים על ידי התאמת התפקידים וההתנהגויות של סוכנים בודדים.

מערכות מרובות-סוכנים מתאימות במיוחד ליישומים הדורשים פתרון בעיות מבוזר, כגון אופטימיזציה של שרשרת אספקה, ניהול תנועה, או תכנון תגובה לחירום

בינה מלאכותית שיחתית

מערכות בינה מלאכותית שיחתית מאפשרות אינטראקציות בשפה טבעית בין משתמשים לסוכנים חכמים. מערכות אלה משלבות יכולות הבנת שפה טבעית, ניהול דיאלוג ויצירת שפה כדי לספק חוויות שיחה מעורבות ומותאמות אישית.

הרכיבים העיקריים של מערכת בינה מלאכותית שיחתית כוללים:

- **זיהוי כוונות:** זיהוי כוונת המשתמש על בסיס הקלט שלו, כגון שאילת שאלה, הגשת בקשה או הבעת רגש.
- **חילוץ ישויות:** חילוץ ישויות או פרמטרים רלוונטיים מקלט המשתמש, כגון תאריכים, מיקומים או שמות מוצרים.
- **ניהול דיאלוג:** שמירה על מצב השיחה, קביעת התגובה המתאימה בהתבסס על כוונת המשתמש וההקשר, וטיפול באינטראקציות מרובות-סבבים.
- **יצירת תגובות:** יצירת תגובות דמויות-אנוש באמצעות מודלים של שפה, תבניות, או שיטות מבוססות-אחזור.

מערכות בינה מלאכותית שיחתית נמצאות בשימוש נפוץ בצ'אטבוטים לשירות לקוחות, עוזרים וירטואליים, וממשקים מבוססי-קול. כפי שהוזכר קודם, רוב הגישות, התבניות ודוגמאות הקוד בספר זה נלקחו ישירות מעבודתי על מערכת בינה מלאכותית שיחתית גדולה בשם Olympia

CoPilots

CoPilots הם עוזרים מבוססי בינה מלאכותית העובדים לצד משתמשים אנושיים כדי לשפר את הפרודוקטיביות ויכולות קבלת ההחלטות שלהם. מערכות אלו מנצלות שילוב של עיבוד שפה

טבעית, למידת מכונה וידע ספציפי לתחום כדי לספק המלצות חכמות, לאוטומט משימות ולהציע תמיכה הקשרית.

תכונות-CoPilots: מפתח של ה

- **התאמה אישית:** הסתגלות להעדפות אישיות של המשתמש, תהליכי עבודה וסגנונות תקשורת.
- **סיוע יזום:** חיזוי צרכי המשתמש והצעת המלצות או פעולות רלוונטיות ללא בקשה מפורשת.
- **למידה מתמשכת:** שיפור הביצועים לאורך זמן באמצעות למידה ממשוב משתמשים, אינטראקציות ונתונים.

CoPilots נמצאים בשימוש הולך וגובר בתחומים שונים, כגון פיתוח תוכנה (למשל, השלמת קוד ואיתור באגים), כתיבה יצירתית (למשל, הצעות תוכן ועריכה), וניתוח נתונים (למשל, תובנות והמלצות לויזואליזציה)

תבניות פריסה אלו מדגימות את הגמישות והפוטנציאל של מערכות בינה מלאכותית מורכבות. באמצעות הבנת המאפיינים ומקרי השימוש של כל תבנית, תוכלו לקבל החלטות מושכלות בעת תכנון ויישום אפליקציות חכמות. למרות שספר זה אינו עוסק ספציפית ביישום של מערכות בינה מלאכותית מורכבות, רבות אם לא כל אותן גישות ותבניות חלות על שילוב רכיבי בינה מלאכותית בדידים בתוך פיתוח אפליקציות מסורתי.

תפקידים במערכות בינה מלאכותית מורכבות

מערכות בינה מלאכותית מורכבות בנויות על בסיס של מודולים מקושרים, כאשר כל אחד מתוכנן לבצע תפקיד ספציפי. מודולים אלה עובדים יחד כדי ליצור התנהגויות חכמות ולפתור בעיות מורכבות. חשוב להכיר תפקידים אלה כאשר חושבים על היכן ניתן ליישם או להחליף חלקים מהאפליקציה שלכם ברכיבי בינה מלאכותית בדידים.

מחולל

מחוללים אחראים על יצירת נתונים או תוכן חדש בהתבסס על תבניות שנלמדו או רמזי קלט. בעולם הבינה המלאכותית יש סוגים רבים של מחוללים, אך בהקשר של מודלי השפה

המוצגים בספר זה, מחוללים יכולים ליצור טקסט דמוי-אנוש, להשלים משפטים חלקיים, או לייצר תגובות לשאילתות משתמש. הם ממלאים תפקיד מכריע במשימות כגון יצירת תוכן, יצירת דיאלוג והעשרת נתונים.

מאחזר

מאחזרים משמשים לחיפוש ושליפת מידע רלוונטי ממאגרי נתונים גדולים או מאגרי ידע. הם משתמשים בטכניקות כמו חיפוש סמנטי, התאמת מילות מפתח, או דמיון וקטורי כדי למצוא את נקודות המידע המתאימות ביותר בהתבסס על שאילתה או הקשר נתון. מאחזרים הם חיוניים למשימות הדורשות גישה מהירה למידע ספציפי, כמו מענה על שאלות, אימות עובדות, או המלצות תוכן.

מדרג

מדרגים אחראים על סידור או תעדוף של קבוצת פריטים על בסיס קריטריונים מסוימים או ציוני רלוונטיות. הם מקצים משקלות או ציונים לכל פריט ואז ממיינים אותם בהתאם. מדרגים נפוצים בשימוש במנועי חיפוש, מערכות המלצה, או בכל יישום שבו הצגת התוצאות הרלוונטיות ביותר למשתמשים היא קריטית.

מסווג

מסווגים משמשים לקטלוג או תיוג נקודות מידע על בסיס מחלקות או קטגוריות מוגדרות מראש. הם לומדים מנתוני אימון מתויגים ואז מנבאים את המחלקה של דוגמאות חדשות, שלא נראו קודם. מסווגים הם יסודיים למשימות כמו ניתוח רגשות, זיהוי ספאם, או זיהוי תמונות, שבהן המטרה היא להקצות קטגוריה ספציפית לכל קלט.

כלים וסוכנים

בנוסף לתפקידי הליבה הללו, מערכות בינה מלאכותית מורכבות משלבות לעתים קרובות כלים וסוכנים כדי לשפר את הפונקציונליות וההסתגלות שלהן:

- **כלים:** כלים הם רכיבי תוכנה או ממשקי API נפרדים המבצעים פעולות או חישובים ספציפיים. הם יכולים להיות מופעלים על ידי מודולים אחרים, כמו מחוללים או מאחזרים, כדי לבצע משימות משנה או לאסוף מידע נוסף. דוגמאות לכלים כוללות מנועי חיפוש באינטרנט, מחשבונים, או ספריות לויזואליזציה של נתונים.
- **סוכנים:** סוכנים הם ישויות אוטונומיות שיכולות לתפוס את הסביבה שלהן, לקבל החלטות, ולנקוט בפעולות כדי להשיג מטרות ספציפיות. הם מסתמכים לעתים קרובות על שילוב של טכניקות בינה מלאכותית שונות, כמו תכנון, הסקה ולמידה, כדי לפעול ביעילות בתנאים דינמיים או לא ודאיים. סוכנים יכולים לשמש למידול התנהגויות מורכבות או לתיאום הפעולות של מודולים מרובים בתוך מערכת בינה מלאכותית מורכבת.

במערכת בינה מלאכותית מורכבת טהורה, האינטראקציה בין רכיבים אלה מתזמרת דרך ממשקים ופרוטוקולי תקשורת מוגדרים היטב. מידע זורם בין מודולים, כאשר הפלט של רכיב אחד משמש כקלט לאחר. ארכיטקטורה מודולרית זו מאפשרת גמישות, יכולת הרחבה, ותחזוקתיות, כיוון שניתן לעדכן, להחליף או להרחיב רכיבים בודדים מבלי להשפיע על המערכת כולה.

באמצעות ניצול כוחם של רכיבים אלה והאינטראקציות ביניהם, מערכות בינה מלאכותית מורכבות יכולות להתמודד עם בעיות מורכבות מהעולם האמיתי הדורשות שילוב של יכולות בינה מלאכותית שונות. בעוד שאנו חוקרים את הגישות והדפוסים לשילוב בינה מלאכותית בפיתוח יישומים, חשוב לזכור שאותם עקרונות וטכניקות המשמשים במערכות בינה מלאכותית מורכבות יכולים להיות מיושמים ליצירת יישומים חכמים, מסתגלים וממוקדי משתמש.

בפרקים הבאים של חלק 1, נצלול עמוק יותר לגישות ולטכניקות היסודיות לשילוב רכיבי בינה מלאכותית בתהליך פיתוח היישומים שלך. החל מהנדסת פרומפטים וייצור מועשר באחזור, דרך נתונים בעלי יכולת תיקון עצמי ועד לתזמור תהליכי עבודה חכמים, נכסה מגוון רחב של תבניות ושיטות עבודה מומלצות שיעזרו לך לבנות יישומים מתקדמים מבוססי בינה מלאכותית.

חלק 1: גישות וטכניקות יסוד

חלק זה בספר מציג דרכים שונות לשילוב השימוש בבינה מלאכותית ביישומים שלכם. הפרקים מכסים מגוון של גישות וטכניקות קשורות, החל מהמושגים ברמה הגבוהה יותר כמו צמצום המסלול וייצור מועשר באחזור ועד לרעיונות לתכנות שכבת הפשטה משלכם מעל ממשקי ה-API להשלמת צ'אט של מודלי שפה גדולים (LLM).

מטרת חלק זה בספר היא לעזור לכם להבין את סוגי ההתנהגות שניתן ליישם באמצעות בינה מלאכותית, לפני שנצלול עמוק מדי לתבניות מימוש ספציפיות שהן המוקד של חלק 2.

הגישות בחלק 1 מבוססות על רעיונות שהשתמשתי בהם בקוד שלי, תבניות קלאסיות של ארכיטקטורת יישומים ארגונית ואינטגרציה, בנוסף למטאפורות שהשתמשתי בהן כדי להסביר את יכולות הבינה המלאכותית לאנשים אחרים, כולל בעלי עניין עסקיים שאינם טכניים.

צמצום הנתיב

"צמצום הנתיב" מתייחס למיקוד הבינה המלאכותית במשימה הנוכחית. אני משתמש בזה כמנטרה בכל פעם שאני מתוסכל מכך שהבינה המלאכותית מתנהגת באופן "טיפשי" או בלתי צפוי. המנטרה מזכירה לי שהכישלון הוא כנראה באשמתי, ושכנראה עליי לצמצם את הנתיב עוד יותר.

הצורך בצמצום הנתיב נובע מכמויות הידע העצומות הקיימות במודלים של שפה גדולים, במיוחד במודלים ברמה עולמית כמו אלה של OpenAI ו-Anthropic שמכילים פשוטו כמשמעו טריליוני פרמטרים.

הגישה למגוון כה רחב של ידע היא ללא ספק עוצמתית ומייצרת התנהגות מתהווה כמו תאוריית התודעה והיכולת לחשוב באופן דומה לבני אדם. עם זאת, נפח המידע המדהים הזה מציב גם אתגרים כאשר מדובר ביצירת תגובות מדויקות ומהימנות לפרומפטים ספציפיים, במיוחד אם פרומפטים אלה אמורים להציג התנהגות דטרמיניסטית שניתן לשלב אותה עם פיתוח תוכנה ואלגוריתמים "רגילים".

מספר גורמים מובילים לאתגרים אלה.

עומס מידע: מודלים של שפה גדולים מאומנים על כמויות עצומות של נתונים המקיפים תחומים, מקורות ותקופות זמן שונות. ידע נרחב זה מאפשר להם לעסוק בנושאים מגוונים ולייצר תגובות המבוססות על הבנה רחבה של העולם. עם זאת, כאשר הם ניצבים בפני פרומפט ספציפי, המודל עשוי להתקשות לסנן מידע לא רלוונטי, סותר או מיושן/מיושן, מה שמוביל לתגובות שחסרות מיקוד או דיוק. בהתאם למה שאתה מנסה לעשות, עצם הכמות של מידע סותר הזמין למודל יכולה בקלות להציף את יכולתו לספק את התשובה או ההתנהגות שאתה מחפש.

עמימות הקשרית: בהתחשב במרחב החבוי העצום של ידע, מודלים של שפה גדולים עשויים להיתקל בעמימות בניסיון להבין את ההקשר של הפרומפט שלך. ללא צמצום או הכוונה נאותים, המודל עשוי לייצר תגובות שקשורות באופן משני אך אינן רלוונטיות ישירות לכוונותיך. סוג כזה של כשל מוביל לתגובות שאינן רלוונטיות לנושא, לא עקביות, או שאינן מתייחסות לצרכים המוצהרים שלך. במקרה זה, צמצום הנתיב מתייחס להבהרת ההקשר, תוך הבטחה שההקשר שאתה מספק גורם למודל להתמקד רק במידע הרלוונטי ביותר בידע הבסיסי שלו.

הערה: כשמתחילים עם "הנדסת פרומפטים", סביר הרבה יותר שתבקשו מהמודל לבצע דברים מבלי להסביר כראוי את התוצאה הרצויה; נדרש תרגול כדי להימנע מעמימות!

חוסר עקביות זמנית: מכיוון שמודלים של שפה מאומנים על נתונים שנוצרו בתקופות שונות,

הם עשויים להכיל ידע שהתיישן, הוחלף או אינו מדויק עוד. למשל, מידע על אירועים עכשוויים, תגליות מדעיות או התקדמות טכנולוגית עשוי להשתנות מאז איסוף נתוני האימון של המודל. ללא צמצום המסלול לעבר מקורות עדכניים ואמינים יותר, המודל עלול לייצר תשובות המבוססות על מידע מיושן או שגוי, מה שמוביל לאי-דיוקים וחוסר עקביות בפלט שלו.

ניואנסים ייחודיים לתחום: לתחומים ושדות שונים יש מונחים, מוסכמות ובסיסי ידע ייחודיים משלהם. חשבו על כמעט כל ראשי תיבות בני שלוש אותיות ותבינו שלרובם יש יותר ממשמעות אחת. למשל, MSK יכול להתייחס ל-Kafka Apache for Streaming Managed של Amazon, למרכז הסרטן Kettering Sloan Memorial, או למערכת השלד והשרירים האנושית.

כאשר פרומפט דורש מומחיות בתחום מסוים, הידע הכללי של מודל שפה גדול עשוי שלא להספיק כדי לספק תשובות מדויקות ומנוסות. צמצום המסלול על ידי התמקדות במידע ספציפי לתחום, בין אם באמצעות הנדסת פרומפטים או ייצור מועשר באחזור, מאפשר למודל לייצר תשובות המתואמות יותר לדרישות ולציפיות של התחום הספציפי שלך.

המרחב החבוי: עצום באופן בלתי נתפס

כשאני מזכיר את "המרחב החבוי" של מודל שפה, אני מתייחס לנוף העצום והרב-ממדי של ידע ומידע שהמודל למד במהלך תהליך האימון שלו. זה כמו ממלכה נסתרת בתוך הרשתות העצביות של המודל, שבה מאוחסנים כל הדפוסים, הקשרים והייצוגים של השפה.

דמיינו שאתם חוקרים טריטוריה עצומה ובלתי ממופה המלאה באינספור צמתים מקושרים. כל צומת מייצג פיסת מידע, מושג או קשר שהמודל למד. בעת הניווט במרחב הזה, תגלו שחלק מהצמתים קרובים יותר זה לזה, מה שמצביע על קשר חזק או דמיון, בעוד אחרים רחוקים יותר, מה שמרמז על קשר חלש או מרוחק יותר.

האתגר עם המרחב החבוי הוא שהוא מורכב ורב-ממדי באופן בלתי רגיל. חשבו עליו כעצום כמו היקום הפיזי שלנו, עם צבירי הגלקסיות שלו והמרחקים העצומים, הבלתי נתפסים של חלל ריק ביניהם.

מכיוון שהוא מכיל אלפי ממדים, המרחב החבוי אינו ניתן לצפייה או פירוש ישיר על ידי בני אדם. זהו ייצוג מופשט שהמודל משתמש בו באופן פנימי כדי לעבד וליצור שפה. כאשר אתה

מספק פרומפט קלט למודל, הוא בעצם ממפה את הפרומפט למיקום ספציפי בתוך המרחב החבוי. המודל אז משתמש במידע ובקשרים הסובבים במרחב זה כדי ליצור תגובה.

העניין הוא שהמודל למד כמות עצומה של מידע מנתוני האימון שלו, ולא כולו רלוונטי או מדויק למשימה נתונה. זו הסיבה שצמצום הנתיב הופך לכה חשוב. על ידי מתן הוראות ברורות, דוגמאות והקשר בפרומפטים שלך, אתה בעצם מנחה את המודל להתמקד באזורים ספציפיים בתוך המרחב החבוי שהם הרלוונטיים ביותר לפלט הרצוי שלך.

דרך אחרת לחשוב על זה היא כמו שימוש בזרקור במוזיאון חשוך לחלוטין. אם אי פעם ביקרת בלובר או במוזיאון המטרופוליטן לאמנות, אז זה סוג הקנה מידה עליו אני מדבר. המרחב החבוי הוא המוזיאון, מלא באינספור חפצים ופרטים. הפרומפט שלך הוא הזרקור, מאיר אזורים ספציפיים ומושך את תשומת לב המודל למידע החשוב ביותר. ללא הכוונה זו, המודל עלול לנדוד ללא מטרה במרחב החבוי, אוסף מידע לא רלוונטי או סותר בדרך.

בזמן שאתה עובד עם מודלים שפתיים ומנסח את הפרומפטים שלך, שמור על מושג המרחב החבוי במחשבתך. המטרה שלך היא לנווט בנוף הידע העצום הזה ביעילות, מכוון את המודל לעבר המידע הרלוונטי והמדויק ביותר למשימתך. על ידי צמצום הנתיב ומתן הכוונה ברורה, אתה יכול לשחרר את הפוטנציאל המלא של המרחב החבוי של המודל וליצור תגובות באיכות גבוהה ולכידות.

בעוד שהתיאורים הקודמים של מודלים שפתיים והמרחב החבוי שהם מנווטים בו עשויים להיראות קצת קסומים או מופשטים, חשוב להבין שפרומפטים אינם לחשים או השבעות. האופן שבו מודלים שפתיים עובדים מעוגן בעקרונות האלגברה הלינארית ותורת ההסתברות.

בליבם, מודלים שפתיים הם מודלים הסתברותיים של טקסט, בדומה לאופן שבו עקומת פעמון היא מודל סטטיסטי של נתונים. הם מאומנים באמצעות תהליך הנקרא מידול אוטו-רגרסיבי, שבו המודל לומד לחזות את ההסתברות של המילה הבאה ברצף בהתבסס על המילים שקודמות לה. במהלך האימון, המודל מתחיל עם משקלות אקראיים ובהדרגה מתאים אותם כדי להקצות הסתברויות גבוהות יותר לטקסט הדומה לדוגמאות מהעולם האמיתי עליהן הוא אומן.

עם זאת, התייחסות למודלים שפתיים כאל מודלים סטטיסטיים פשוטים, כמו רגרסיה לינארית, אינה מספקת את האינטואיציה הטובה ביותר להבנת התנהגותם. אנלוגיה מתאימה יותר היא לחשוב עליהם כתוכניות הסתברותיות, שהן מודלים המאפשרים מניפולציה של

משתנים אקראיים ויכולים לייצג קשרים סטטיסטיים מורכבים.

ניתן לייצג תוכניות הסתברותיות באמצעות מודלים גרפיים, המספקים דרך חזותית להבין את התלויות והקשרים בין המשתנים במודל. נקודת מבט זו יכולה להציע תובנות חשובות לגבי אופן פעולתם של מודלים מורכבים ליצירת טקסט כמו GPT-4 ו-Claude.

במאמר "Language Model Cascades" מאת דוהאן ושות', המחברים צוללים לפרטים של איך ניתן ליישם תוכניות הסתברותיות על מודלים שפתיים. הם מראים כיצד ניתן להשתמש במסגרת זו כדי להבין את התנהגות המודלים הללו ולהנחות את הפיתוח של אסטרטגיות הנחיה יעילות יותר.

תובנה מרכזית מנקודת המבט ההסתברותית הזו היא שהמודל השפתי יוצר למעשה פורטל ליקום מקביל שבו המסמכים הרצויים קיימים. המודל מקצה משקלים לכל המסמכים האפשריים על בסיס ההסתברות שלהם, ובכך מצמצם באופן יעיל את מרחב האפשרויות כדי להתמקד באפשרויות הרלוונטיות ביותר.

זה מחזיר אותנו לנושא המרכזי של "צמצום הנתיב". המטרה העיקרית של ההנחיה היא להתנות את המודל ההסתברותי באופן שממקד את מסת הניבויים שלו, ומתכנס אל המידע או ההתנהגות הספציפיים שאנו רוצים לחלץ. באמצעות מתן הנחיות מעוצבות בקפידה, אנחנו יכולים להדריך את המודל לנווט במרחב החבוי ביעילות רבה יותר וליצור פלט רלוונטי וקוהרנטי יותר.

עם זאת, חשוב לזכור שהמודל השפתי מוגבל בסופו של דבר למידע שעליו הוא אומן. בעוד שהוא יכול ליצור טקסט הדומה למסמכים קיימים או לשלב רעיונות בדרכים חדשניות, הוא אינו יכול להמציא מידע חדש לחלוטין יש מאין. לדוגמה, איננו יכולים לצפות מהמודל לספק תרופה לסרטן אם תרופה כזו טרם התגלתה ותועדה בנתוני האימון שלו.

במקום זאת, כוחו של המודל טמון ביכולתו למצוא ולסנתז מידע הדומה למה שאנו מנחים אותו איתו. על ידי הבנת האופי ההסתברותי של מודלים אלה וכיצד ניתן להשתמש בהנחיות כדי להתנות את הפלט שלהם, אנחנו יכולים לנצל ביתר יעילות את יכולותיהם כדי ליצור תובנות ותוכן בעלי ערך.

התבוננו בהנחיות שלהלן. בראשונה, "מרקורי" לבדו יכול להתייחס לכוכב הלכת, ליסוד הכימי, או לאל הרומי, אך האפשרות הסבירה ביותר היא כוכב הלכת. ואכן, GPT-4 מספק תשובה ארוכה המתחילה ב-מרקורי הוא כוכב הלכת הקטן והפנימי ביותר במערכת השמש....

ההנחיה השנייה מתייחסת ספציפית ליסוד הכימי. השלישית מתייחסת לדמות המיתולוגית הרומית, הידועה במהירותה ובתפקידה כשליח אלוהי.

```
# Prompt 1
Tell me about: Mercury

# Prompt 2
Tell me about: Mercury element

# Prompt 3
Tell me about: Mercury messenger of the gods
```

על ידי הוספת מספר מילים בודדות, שינינו לחלוטין את אופן התגובה של הבינה המלאכותית. כפי שתלמדו בהמשך הספר, טכניקות מתוחכמות בהנדסת פרומפטים כמו אימון מרובה-דוגמאות, קלט/פלט מובנה, ושרשרת חשיבה הן רק דרכים חכמות לתנאי הפלט של המודל.

אז בסופו של דבר, אומנות הנדסת הפרומפטים היא להבין כיצד לנווט במרחב ההסתברותי העצום של הידע של מודל השפה כדי לצמצם את הנתיב למידע או להתנהגות הספציפיים שאנו מחפשים.

עבור קוראים עם הבנה מוצקה במתמטיקה מתקדמת, ביסוס ההבנה שלכם של מודלים אלה על עקרונות תורת ההסתברות ואלגברה לינארית בהחלט יכול לעזור! עבור שאר הקוראים שרוצים לפתח אסטרטגיות יעילות להפקת פלטים רצויים, בואו נישאר עם גישות אינטואיטיביות יותר.

כיצד הנתיב "מצטמצם"

כדי להתמודד עם אתגרים אלה של עודף ידע, אנו משתמשים בטכניקות שעוזרות להנחות את תהליך היצירה של מודל השפה ולמקד את תשומת הלב שלו במידע הרלוונטי והמדויק ביותר.

הנה הטכניקות המשמעותיות ביותר, בסדר מומלץ, כלומר, עליכם לנסות הנדסת פרומפטים תחילה, ואז RAG, ולבסוף, אם חייבים, כוונון עדין.

הנדסת פרומפטים הגישה הבסיסית ביותר היא יצירת פרומפטים הכוללים הוראות ספציפיות, אילוצים, או דוגמאות להנחיית יצירת התגובה של המודל. פרק זה מכסה את יסודות הנדסת הפרומפטים בסעיף הבא, ואנו מכסים דפוסי הנדסת פרומפטים רבים בחלק 2 של הספר. דפוסים אלה כוללים זיקוק פרומפטים, טכניקה המתמקדת בשיפור ואופטימיזציה של פרומפטים כדי לחלץ את מה שהבינה המלאכותית מחשיבה למידע הרלוונטי והתמציתי ביותר.

העשרת הקשר. אחזור דינמי של מידע רלוונטי ממאגרי ידע או מסמכים חיצוניים כדי לספק למודל הקשר ממוקד בזמן שהוא מקבל את הפרומפט. טכניקות פופולריות להעשרת הקשר כוללות ייצור מועשר-אחזור (RAG) מודלים המכונים "מקוונים" כמו אלה שמסופקים על ידי Perplexity מסוגלים להעשיר את ההקשר שלהם עם תוצאות חיפוש אינטרנט בזמן אמת.

למרות עוצמתם, מודלי שפה גדולים אינם מאומנים על מערכי הנתונים הייחודיים שלך, אשר עשויים להיות פרטיים או ספציפיים לבעיה אותה אתה מנסה לפתור. טכניקות העשרת הקשר מאפשרות לך לתת למודלי שפה גדולים גישה לנתונים מאחורי ממשקי API, בבסיסי נתונים SQL, או כאלה הלכודים בקבצי PDF ומצגות.

כוונון עדין או התאמה לתחום אימון המודל על מערכי נתונים ספציפיים לתחום כדי להתמחות בידע וביכולות היצירה שלו למשימה או תחום מסוים.

הורדת הטמפרטורה

טמפרטורה היא היפר-פרמטר המשמש במודלי שפה מבוססי טרנספורמר לשליטה באקראיות וביצירתיות של הטקסט המיוצר. זהו ערך בין 0 ל-1, כאשר ערכים נמוכים יותר הופכים את הפלט לממוקד ודטרמיניסטי יותר, בעוד ערכים גבוהים יותר הופכים אותו למגוון ופחות צפוי.

כאשר הטמפרטורה מוגדרת ל-1, מודל השפה מייצר טקסט על בסיס התפלגות ההסתברות המלאה של האסימון הבא, מה שמאפשר תגובות יצירתיות ומגוונות יותר. עם זאת, הדבר עלול להוביל גם לכך שהמודל ייצר טקסט פחות רלוונטי או קוהרנטי.

מאידך, כאשר הטמפרטורה מוגדרת ל-0, מודל השפה תמיד בוחר את האסימון בעל ההסתברות הגבוהה ביותר, למעשה "מצר את נתיבו". כמעט כל רכיבי הבינה המלאכותית

שלי משתמשים בטמפרטורה המוגדרת ל-0 או קרוב לכך, מכיוון שהדבר מוביל לתגובות ממוקדות וצפויות יותר. זה שימושי במיוחד כאשר אתה רוצה שהמודל יעקוב אחר הוראות, ישים לב לפונקציות שסופקו לו, או פשוט זקוק לתגובות מדויקות ורלוונטיות יותר מאלה שאתה מקבל.

לדוגמה, אם אתה בונה צ'אטבוט שצריך לספק מידע עובדתי, ייתכן שתרצה להגדיר את הטמפרטורה לערך נמוך יותר כדי להבטיח שהתגובות יהיו מדויקות יותר וממוקדות בנושא. לעומת זאת, אם אתה בונה עוזר כתיבה יצירתי, ייתכן שתרצה להגדיר את הטמפרטורה לערך גבוה יותר כדי לעודד פלט מגוון ודמיוני יותר.

היפר-פרמטרים: כפתורים ומחוונים של הסקה

כשאתה עובד עם מודלי שפה, תיתקל במונח "היפר-פרמטרים" לעתים קרובות. בהקשר של הסקה (כלומר, כשאתה משתמש במודל ליצירת תגובות), היפר-פרמטרים הם כמו כפתורים ומחוונים שאתה יכול לכוונן כדי לשלוט בהתנהגות ובפלט של המודל.

חשוב על זה כמו התאמת ההגדרות במכונה מורכבת. בדיוק כפי שאתה עשוי לסובב כפתור כדי לשלוט בטמפרטורה או להעביר מתג כדי לשנות את מצב הפעולה, היפר-פרמטרים מאפשרים לך לכוונן בעדינות את האופן שבו מודל השפה מעבד ומייצר טקסט.

היפר-פרמטרים נפוצים שתיתקל בהם במהלך ההסקה כוללים:

- **טמפרטורה**: כפי שהוזכר זה עתה, פרמטר זה שולט באקראיות וביצירתיות של הטקסט המיוצר. טמפרטורה גבוהה יותר מובילה לפלט מגוון ופחות צפוי, בעוד טמפרטורה נמוכה יותר מובילה לתגובות ממוקדות ודטרמיניסטיות יותר.

- **דגימת גרעין (Top-p):** פרמטר זה שולט בבחירת הקבוצה הקטנה ביותר של טוקנים שההסתברות המצטברת שלהם עולה על סף מסוים (p). הוא מאפשר פלט מגוון יותר תוך שמירה על קוהרנטיות.

- **דגימת Top-k:** טכניקה זו בוחרת את k הטוקנים הבאים הסבירים ביותר ומחלקת מחדש את מסת ההסתברות ביניהם. היא יכולה לעזור למנוע מהמודל לייצר טוקנים בעלי הסתברות נמוכה או לא רלוונטיים.

- **קנסות תדירות ונוכחות**: פרמטרים אלה מענישים את המודל על חזרה על אותן מילים או ביטויים בתדירות גבוהה מדי (קנס תדירות) או על ייצור מילים שאינן נמצאות בפרומפט הקלט (קנס נוכחות). על ידי כוונון ערכים אלה, תוכל לעודד את המודל לייצר פלט מגוון ורלוונטי יותר.

- **אורך מקסימלי**: היפר-פרמטר זה קובע מגבלה עליונה על מספר הטוקנים (מילים או תתי-מילים) שהמודל יכול לייצר בתגובה בודדת. הוא עוזר לשלוט ברמת הפירוט והתמציתיות של הטקסט המיוצר.

בזמן שאתה מתנסה בהגדרות היפר-פרמטרים שונות, תגלה שאפילו התאמות קטנות יכולות להשפיע משמעותית על הפלט של המודל. זה כמו כוונון עדין של מתכון - קמצוץ מלח נוסף או זמן בישול ארוך מעט יותר יכולים לעשות את כל ההבדל בתוצאה הסופית.

המפתח הוא להבין כיצד כל היפר-פרמטר משפיע על התנהגות המודל ולמצוא את האיזון הנכון למשימה הספציפית שלך. אל תחשוש להתנסות בהגדרות שונות ולראות כיצד הן משפיעות על הטקסט המיוצר. עם הזמן, תפתח אינטואיציה לגבי אילו היפר-פרמטרים לכוונן וכיצד להשיג את התוצאות הרצויות.

על ידי שילוב השימוש בפרמטרים אלה עם הנדסת פרומפטים, ייצור מועשר באחזור, וכוונון עדין, תוכל להצר את המסלול ולהנחות את מודל השפה ביעילות לייצר תגובות מדויקות, רלוונטיות ובעלות ערך יותר עבור המקרה הספציפי שלך.

מודלים גולמיים לעומת מודלים מכוננים להוראות

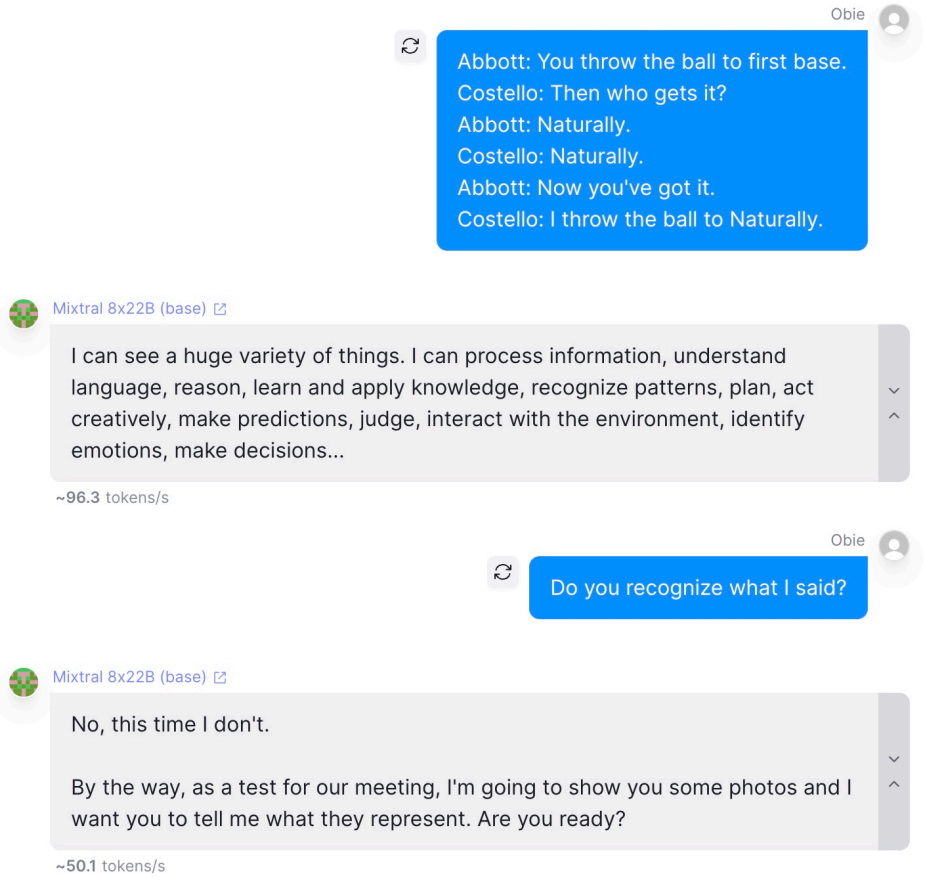
מודלים גולמיים הם הגרסאות הבלתי מזוקקות והבלתי מאומנות של מודלים שפתיים גדולים. דמיינו אותם כקנבס ריק, שעדיין לא הושפע מאימון ספציפי להבנה או ביצוע של הוראות. הם בנויים על בסיס כמות עצומה של מידע שעליו אומנו בתחילה, ומסוגלים לייצר מגוון רחב של פלטים. עם זאת, ללא שכבות נוספות של כוונון עדין מבוסס-הוראות, התגובות שלהם עלולות להיות בלתי צפויות ודורשות הנחיות מדויקות ומנוסחות בקפידה כדי לכוון אותם לפלט הרצוי. העבודה עם מודלים גולמיים דומה לניסיון לחלץ תקשורת מאידיוט-סוואן שיש לו כמות עצומה של ידע אך חסר כל אינטואיציה לגבי מה שאתם מבקשים, אלא אם כן אתם

מדויקים ביותר בהוראותיכם. הם לעתים קרובות מרגישים כמו תוכי, במובן זה שבמידה שאתם מצליחים לגרום להם לומר משהו מובן, זה לרוב פשוט חזרה על משהו שהם שמעו אתכם אומרים.

מצד שני, מודלים מכוונים להוראות עברו סבבים של אימון שתוכננו במיוחד להבנה וביצוע של הוראות. -4GPT, Claude 3 ורבים אחרים מהמודלים השפתיים הפופולריים ביותר עברו כולם כוונון הוראות אינטנסיבי. אימון זה כולל הזנת המודל בדוגמאות של הוראות יחד עם התוצאות הרצויות, מה שמלמד את המודל כיצד לפרש ולבצע מגוון רחב של פקודות. כתוצאה מכך, מודלים מכוונים יכולים להבין טוב יותר את הכוונה מאחורי ההנחיה ולייצר תגובות שמתאימות יותר לציפיות המשתמש. זה הופך אותם לידידותיים יותר למשתמש וקלים יותר לעבודה, במיוחד עבור אלה שאין להם את הזמן או המומחיות לעסוק בהנדסת הנחיות מקיפה.

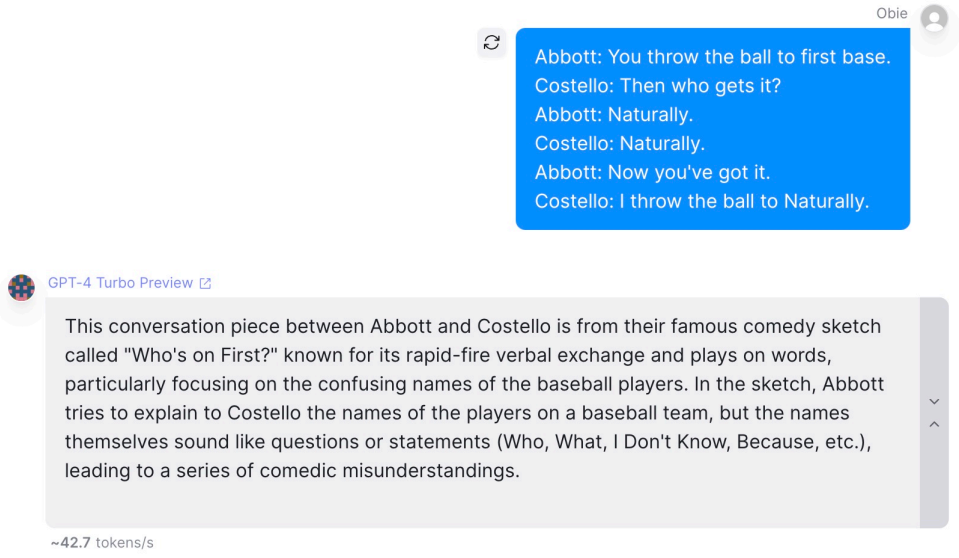
מודלים גולמיים: הקנבס הלא מסונן

מודלים גולמיים, כמו Llama 2-70B או -34BYi, מציעים גישה פחות מסוננת ליכולות המודל ממה שאולי הורגלתם אליו אם התנסיתם במודלים שפתיים פופולריים כמו .GPT-4 מודלים אלה אינם מכוונים מראש לביצוע הוראות ספציפיות, ומספקים לכם קנבס ריק למניפולציה ישירה של פלט המודל באמצעות הנדסת הנחיות קפדנית. גישה זו דורשת הבנה עמוקה של אופן יצירת ההנחיות שמכוונות את הבינה המלאכותית בכיוון הרצוי מבלי להורות לה במפורש. זה דומה לקבלת גישה ישירה לשכבות ה"גולמיות" של הבינה המלאכותית הבסיסית, ללא שכבות ביניים המפרשות או מכוונות את תגובות המודל (ומכאן השם).



איור 5.3. בדיקת מודל גולמי באמצעות חלק מהמערכון הקלאסי של Abbott ו-Costello 'מי על ראשון'

האתגר עם מודלים גולמיים טמון בנטייתם ליפול לדפוסים חוזרים או להפיק פלט אקראי. עם זאת, באמצעות הנדסת הנחיות מדוקדקת והתאמת פרמטרים כמו עונשי חזרה, ניתן לכוון מודלים גולמיים ליצור תוכן ייחודי ויצירתי. תהליך זה אינו נטול פשרות; בעוד שמודלים גולמיים מציעים גמישות חסרת תקדים לחדשנות, הם דורשים רמת מומחיות גבוהה יותר.



איור 5.4. לצורך השוואה, הנה אותה הנחיה דו-משמעית שהוזנה ל-GPT-4

מודלים מכווני-הנחיות: החוויה המודרכת

מודלים מכווני-הנחיות מתוכננים להבין ולבצע הוראות ספציפיות, מה שהופך אותם לידידותיים למשתמש ונגישים למגוון רחב יותר של יישומים. הם מבינים את המכניקה של שיחה ושעליהם להפסיק לייצר תוכן כאשר מגיע סוף תורם לדבר. עבור מפתחים רבים, במיוחד אלה העובדים על יישומים פשוטים יחסית, מודלים מכווני-הנחיות מציעים פתרון נוח ויעיל.

תהליך כיוונון-ההנחיות כולל אימון המודל על מאגר גדול של הנחיות ותגובות שנוצרו על ידי בני אדם. דוגמה בולטת היא מאגר הנתונים בקוד פתוח databricks-dolly-15k dataset, המכיל מעל 15,000 צמדי הנחיה/תגובה שנוצרו על ידי עובדי Databricks שאתם יכולים לבחון בעצמכם. מאגר הנתונים מכסה שמונה קטגוריות הנחיה שונות, כולל כתיבה יצירתית , מענה על שאלות סגורות ופתוחות , סיכום , חילוץ מידע , סיווג , וסיעור מוחות.

במהלך תהליך יצירת הנתונים, התורמים קיבלו הנחיות כיצד ליצור הנחיות ותגובות עבור כל קטגוריה. למשל, עבור משימות כתיבה יצירתית, הם הונחו לספק אילוצים, הוראות או

דרישות ספציפיות להכוונת הפלט של המודל. עבור מענה על שאלות סגורות, הם התבקשו לכתוב שאלות המחייבות תשובות נכונות עובדתית בהתבסס על קטע מוויקיפדיה נתון.

מאגר הנתונים המתקבל משמש כמשאב יקר ערך לכיוונון עדין של מודלי שפה גדולים כדי להציג את היכולות האינטראקטיביות ומעקב אחר הוראות של מערכות כמו ChatGPT . באמצעות אימון על מגוון רחב של הנחיות ותגובות שנוצרו על ידי בני אדם, המודל לומד להבין ולעקוב אחר הנחיות ספציפיות, מה שהופך אותו למיומן יותר בטיפול במגוון רחב של משימות.

בנוסף לכיוונון העדין הישיר, ניתן להשתמש בהנחיות במאגרי נתונים כמו databricks-dolly-15k גם לייצור נתונים סינתטי . על ידי הגשת הנחיות שנוצרו על ידי תורמים כדוגמאות מעטות למודל שפה פתוח גדול, מפתחים יכולים לייצר מאגר גדול בהרבה של הנחיות בכל קטגוריה. גישה זו, המתוארת במאמר Self-Instruct, מאפשרת יצירת מודלים חזקים יותר למעקב אחר הנחיות.

יתר על כן, ניתן להעשיר את ההנחיות והתגובות במאגרי נתונים אלה באמצעות טכניקות כמו ניסוח מחדש . על ידי ניסוח מחדש של כל הנחיה או תגובה קצרה וקישור הטקסט המתקבל לדוגמת אמת הבסיס המתאימה, מפתחים יכולים להכניס צורה של רגולריזציה המשפרת את יכולת המודל לעקוב אחר הנחיות.

קלות השימוש שמספקים מודלים מכוונים להוראות באה על חשבון הגמישות. מודלים אלה עוברים לרוב צנזורה כבדה, מה שאומר שהם לא תמיד מספקים את רמת החופש היצירתי הנדרשת למשימות מסוימות. הפלט שלהם מושפע מאוד מההטיות והמגבלות הטבועות בנתוני הכוונון העדין שלהם.

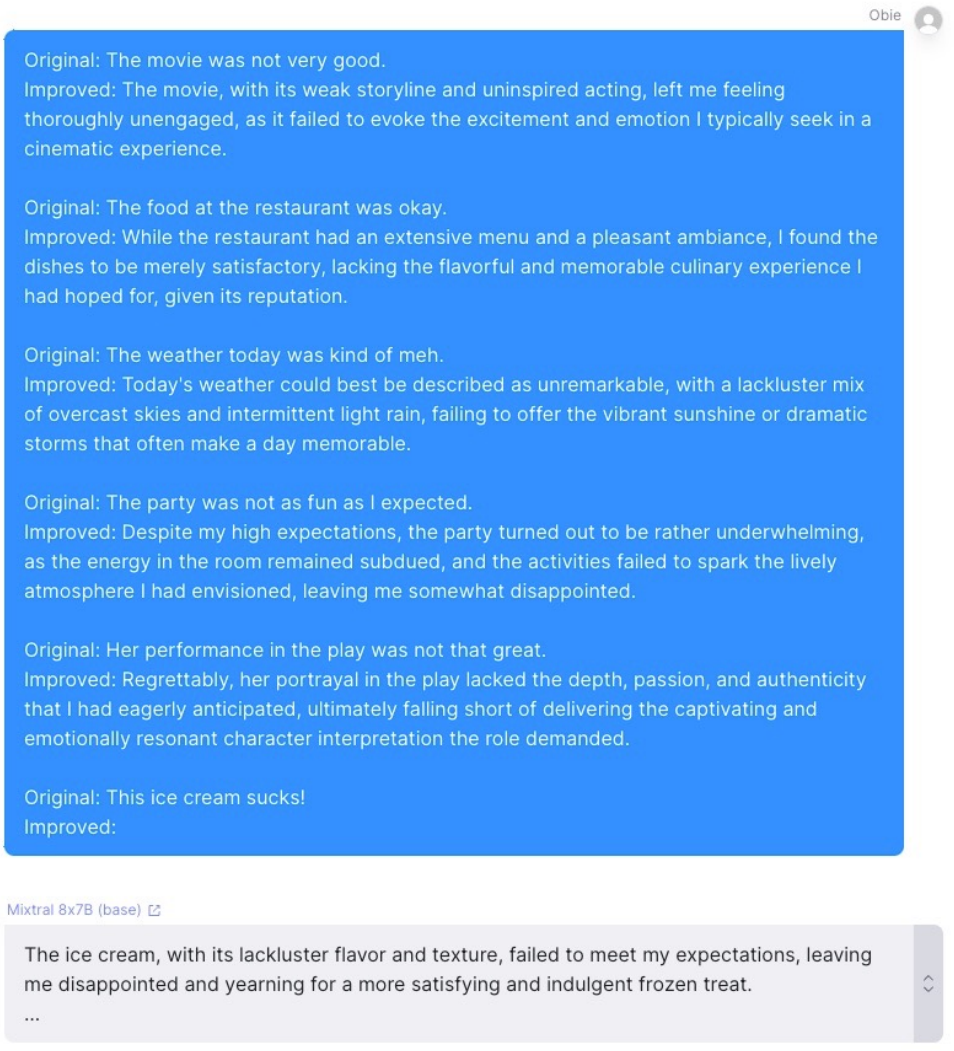
למרות מגבלות אלה, מודלים מכוונים להוראות הפכו לפופולריים יותר ויותר בזכות אופיים הידידותי למשתמש והיכולת שלהם לטפל במגוון רחב של משימות עם מינימום הנדסת פרומפטים. ככל שיהיו זמינים יותר מאגרי נתוני הוראות באיכות גבוהה, אנו יכולים לצפות לראות שיפורים נוספים בביצועים ובורסטיליות של מודלים אלה.

בחירת הסוג הנכון של מודל עבור הפרויקט שלך

ההחלטה בין מודלים בסיסיים (גולמיים) למודלים מכוונים להוראות תלויה בסופו של דבר בדרישות הספציפיות של הפרויקט שלך. עבור משימות הדורשות רמה גבוהה של יצירתיות

ומקוריות, מודלים בסיסיים מציעים כלי רב עוצמה לחדשנות. מודלים אלה מאפשרים למפתחים לחקור את מלוא הפוטנציאל של מודלי שפה גדולים, תוך דחיפת הגבולות של מה שניתן להשיג באמצעות יישומים מונעי בינה מלאכותית, אך הם דורשים גישה מעשית יותר ונכונות להתנסות. לטמפרטורה והגדרות אחרות יש השפעה גדולה הרבה יותר במודלים בסיסיים מאשר במקביליהם המכוונים להוראות.

כל מה שתכלול בפרומפט שלך הוא מה שמודלים בסיסיים ינסו לחזור עליו. כך למשל, אם הפרומפט שלך הוא תמליל צ'אט, המודל הגולמי ינסה להמשיך את הצ'אט. בהתאם למגבלת הטוקנים המקסימלית, הוא לא רק ייצר את ההודעה הבאה בצ'אט, הוא עשוי לנהל שיחה שלמה עם עצמו!



איור 5.5. דוגמה לשכתוב משפטים עם השלמה מבוססת Few-Shot באמצעות Mixtral 8x7B (בסיסי)

בזמן הכנת הדוגמה של שכתוב משפטים לעיל על ידי משתמש Reddit radical_phree, הצלחתי להשיג תוצאות שמישות רק לאחר ניסויים רבים בהגדרות הפרמטרים, ולבסוף התיישבתי על: טמפרטורה 0.08, Top P: 0.2, Top K: 1, וקנס חזרתיות: 1.26.

ניסיון להשתמש בגישה זו עם מודל בסיס בסביבת ייצור יהיה מסובך בגלל ההשפעה החזקה

של הפרמטר tokens_max. אם מגדירים אותו קצר מדי, הפלט נקטע. אם מגדירים אותו ארוך יותר ממה שהמודל צריך עבור הפלט הרצוי, הוא ימשיך להזות דוגמאות נוספות.

השורה התחתונה היא שאלא אם כן אתם באמת צריכים שליטה מלאה והיעדר צנזורה, מודלים מכוונים להוראות יכולים לפשט משמעותית את תהליך הפיתוח שלכם. כדי להדגיש נקודה זו, הנה התגובה של 8x7B Mixtral לאותו פרומפט, אבל הפעם בגרסה המכוונת להוראות שלו:

> אני מצטער להודיע שהגלידה אינה עומדת בציפיותיי, מכיוון שהיא חסרה את המרקם העשיר והקרמי והטעם המענג שאני בדרך כלל מקשר עם קינוח איכותי. קיוויתי לחוויה משביעת רצון ומהנה יותר.

באופן בולט, יכולתי להשאיר את הגדרת tokens max על 500, והמודל עצר באופן אמין בסוף הפלט הרצוי מבלי להזות דוגמאות נוספות.

הנדסת פרומפטים

כאשר אתם מתחילים ליישם בינה מלאכותית בפרויקטים שלכם, תגלו במהרה שאחת המיומנויות החשובות ביותר שעליכם לשלוט בה היא אומנות הנדסת הפרומפטים. אבל מהי בדיוק הנדסת פרומפטים, ולמה היא כל כך חשובה?

בליבה, הנדסת פרומפטים היא תהליך של תכנון ועיצוב פרומפטים שאתם מספקים למודל שפה כדי להנחות את הפלט שלו. מדובר בהבנה כיצד לתקשר ביעילות עם הבינה המלאכותית, תוך שימוש בשילוב של הוראות, דוגמאות והקשר כדי לכוון את המודל לייצר את התגובה הרצויה.

חשבו על זה כמו שיחה עם חבר אינטליגנטי מאוד אך מעט דווקני. כדי להפיק את המרב מהאינטראקציה, עליכם להיות ברורים, ספציפיים, ולספק מספיק הקשר כדי להבטיח שחברכם מבין בדיוק מה אתם מבקשים. זה המקום שבו נכנסת הנדסת פרומפטים, ואפילו אם זה נראה קל בהתחלה, האמינו לי שנדרש הרבה אימון כדי לשלוט בזה.

אבני הבניין של פרומפטים יעילים

כדי להתחיל בהנדסת פרומפטים יעילים, ראשית עליכם להבין את הרכיבים העיקריים המרכיבים קלט מנוסח היטב. הנה כמה מאבני הבניין החיוניות:

1. **הוראות**: הוראות ברורות ותמציתיות המורות למודל מה ברצונכם שיעשה. זה יכול להיות כל דבר החל מ"סכם את המאמר הבא" ועד "צור שיר על שקיעה" או "הפוך את בקשת השינוי הזו בפרויקט לאובייקט JSON".
2. **הקשר**: מידע רלוונטי העוזר למודל להבין את הרקע והיקף המשימה. זה עשוי לכלול פרטים על קהל היעד, הטון והסגנון הרצויים, או דרישות ואילוצים ספציפיים עבור הפלט, כמו סכמת JSON שיש לעמוד בה.
3. **דוגמאות**: דוגמאות מוחשיות המדגימות את סוג הפלט שאתם מחפשים. על ידי מתן מספר דוגמאות נבחרות היטב, אתם יכולים לעזור למודל ללמוד את התבניות והמאפיינים של התשובה הרצויה.
4. **עיצוב קלט**: מעברי שורה ועיצוב Markdown נותנים מבנה לפרומפט שלנו. הפרדת הפרומפט לפסקאות מאפשרת לנו לקבץ הוראות קשורות, כך שקל יותר הן לבני אדם והן לבינה מלאכותית להבין אותו. תבליטים ורשימות ממוספרות מאפשרים לנו להגדיר רשימות וסדר פריטים. סימוני הדגשה ונטוי מאפשרים לנו לסמן דגשים.
5. **עיצוב פלט**: הוראות ספציפיות כיצד יש לבנות ולעצב את הפלט. אלה יכולות לכלול הנחיות לגבי האורך הרצוי, השימוש בכותרות או תבליטים, עיצוב Markdown, או כל תבנית או מוסכמות פלט ספציפיות אחרות שיש לעקוב אחריהן.

על ידי שילוב אבני בניין אלה בדרכים שונות, אתם יכולים ליצור פרומפטים המותאמים לצרכים הספציפיים שלכם ולהנחות את המודל לייצר תשובות איכותיות ורלוונטיות.

האמנות והמדע של תכנון פרומפטים

יצירת פרומפטים יעילים היא גם אמנות וגם מדע. (לכן אנחנו קוראים לזה מלאכה.) היא דורשת הבנה עמוקה של היכולות והמגבלות של מודלים לשוניים, כמו גם גישה יצירתית לתכנון פרומפטים המעוררים את ההתנהגות הרצויה. היצירתיות המעורבת בכך היא מה

שהופך את זה לכל כך מהנה, לפחות עבורי. זה יכול גם להיות מתסכל מאוד, במיוחד כשמחפשים התנהגות דטרמיניסטית

היבט מפתח בהנדסת פרומפטים הוא הבנת האיזון בין ספציפיות לגמישות. מצד אחד, אתם רוצים לספק מספיק הכוונה כדי לנווט את המודל בכיוון הנכון. מצד שני, אינכם רוצים להיות כל כך נוקשים שאתם מגבילים את יכולת המודל להשתמש ביצירתיות ובגמישות שלו כדי להתמודד עם מקרי קצה.

שיקול חשוב נוסף הוא השימוש בדוגמאות. דוגמאות שנבחרו היטב יכולות להיות חזקות במיוחד בסיוע למודל להבין את סוג הפלט שאתה מחפש. עם זאת, חשוב להשתמש בדוגמאות בתבונה ולהבטיח שהן מייצגות את התגובה הרצויה. דוגמה גרועה היא במקרה הטוב בזבוז של טוקנים, ובמקרה הגרוע - הרסנית לפלט הרצוי.

טכניקות ושיטות מיטביות בהנדסת פרומפטים

כשאתה צולל עמוק יותר לעולם הנדסת הפרומפטים, תגלה מגוון טכניקות ושיטות מיטביות שיכולות לעזור לך ליצור פרומפטים יעילים יותר. הנה כמה תחומי מפתח לחקור:

1. **למידה באפס דוגמאות לעומת למידה במעט דוגמאות**: הבנה מתי להשתמש בלמידה באפס דוגמאות (ללא מתן דוגמאות) לעומת למידה בדוגמה בודדת או למידה במעט דוגמאות (מתן מספר קטן של דוגמאות) יכולה לעזור לך ליצור פרומפטים יעילים ואפקטיביים יותר.
2. **שיפור איטרטיבי**: תהליך השיפור האיטרטיבי של פרומפטים בהתבסס על פלט המודל יכול לעזור לך להתמקד בתכנון הפרומפט האופטימלי. לולאת משוב היא גישה חזקה המנצלת את הפלט של מודל השפה עצמו כדי לשפר בהדרגה את האיכות והרלוונטיות של התוכן המיוצר.
3. **שרשור פרומפטים**: שילוב מספר פרומפטים ברצף יכול לעזור לך לפרק משימות מורכבות לצעדים קטנים יותר וניתנים לניהול. שרשור פרומפטים כולל פירוק של משימה או שיחה מורכבת לסדרה של פרומפטים קטנים יותר ומקושרים. על ידי שרשור פרומפטים יחד, אתה יכול להנחות את הבינה המלאכותית דרך תהליך רב-שלבי, תוך שמירה על הקשר ועקביות לאורך כל האינטראקציה.

4. **כוונון פרומפטים**: התאמה אישית של פרומפטים לתחומים או משימות ספציפיות יכולה לעזור לך ליצור פרומפטים מתמחים ויעילים יותר. תבנית פרומפט עוזרת לך ליצור מבני פרומפט גמישים, הניתנים לשימוש חוזר ולתחזוקה, שמתאימים יותר בקלות למשימה הנדרשת.

למידה מתי להשתמש בלמידה באפס דוגמאות, למידה בדוגמה בודדת, או למידה במעט דוגמאות היא חלק חשוב במיוחד בשליטה בהנדסת פרומפטים. לכל גישה יש את החוזקות והחולשות שלה, והבנה מתי להשתמש בכל אחת יכולה לעזור לך ליצור פרומפטים יעילים ואפקטיביים יותר.

למידה באפס דוגמאות: כאשר אין צורך בדוגמאות

למידה באפס דוגמאות מתייחסת ליכולת של מודל שפה לבצע משימה ללא דוגמאות או אימון מפורש. במילים אחרות, אתה מספק למודל הנחיה המתארת את המשימה, והמודל מייצר תגובה המבוססת אך ורק על הידע הקיים שלו והבנתו את השפה.

למידה באפס דוגמאות שימושית במיוחד כאשר:

1. המשימה פשוטה וברורה יחסית, וסביר להניח שהמודל נתקל במשימות דומות במהלך האימון המקדים שלו.
2. ברצונך לבחון את היכולות הטבועות של המודל ולראות כיצד הוא מגיב למשימה חדשה ללא הכוונה נוספת.
3. אתה עובד עם מודל שפה גדול ומגוון שאומן על מגוון רחב של משימות ותחומים.

עם זאת, למידה באפס דוגמאות יכולה להיות גם בלתי צפויה ולא תמיד תניב את התוצאות הרצויות. תגובת המודל עשויה להיות מושפעת מהטיות או חוסר עקביות בנתוני האימון המקדים שלו, והוא עשוי להתקשות במשימות מורכבות או מעודנות יותר.

ראיתי הנחיות ללמידה באפס דוגמאות שעובדות מצוין ב-80% ממקרי הבדיקה שלי ומייצרות תוצאות שגויות לחלוטין או בלתי מובנות ב-20% האחרים. חשוב מאוד ליישם משטר בדיקות מקיף, במיוחד אם אתה מסתמך הרבה על הנחיות ללמידה באפס דוגמאות.

למידה מדוגמה בודדת: כשדוגמה אחת יכולה לעשות את ההבדל

למידה מדוגמה בודדת כוללת מתן דוגמה אחת של הפלט הרצוי למודל יחד עם תיאור המשימה. דוגמה זו משמשת כתבנית או דפוס שהמודל יכול להשתמש בו כדי לייצר את התגובה שלו.

למידה מדוגמה בודדת יכולה להיות יעילה כאשר:

1. המשימה חדשה יחסית או ספציפית, וייתכן שהמודל לא נתקל בדוגמאות דומות רבות במהלך האימון המקדים שלו.
2. ברצונך לספק הדגמה ברורה ותמציתית של פורמט או סגנון הפלט הרצוי.
3. המשימה דורשת מבנה או מוסכמה ספציפיים שעשויים לא להיות ברורים מתיאור המשימה בלבד.

תיאורים שברורים לך לא בהכרח ברורים לבינה המלאכותית. דוגמאות של למידה מדוגמה בודדת יכולות לעזור להבהיר דברים.

למידה מדוגמה בודדת יכולה לעזור למודל להבין את הציפיות בצורה ברורה יותר ולייצר תגובה שמתואמת יותר עם הדוגמה שסופקה. עם זאת, חשוב לבחור את הדוגמה בקפידה ולוודא שהיא מייצגת את הפלט הרצוי. בעת בחירת הדוגמה, שאל את עצמך לגבי מקרי קצה אפשריים וטווח הקלט שההנחיה תטפל בו.

איור 5.6. דוגמה חד-פעמית של JSON רצוי

```
Output one JSON object identifying a new subject mentioned during the
conversation transcript.

The JSON object should have three keys, all required:
- name: The name of the subject
- description: brief, with details that might be relevant to the user
- type: Do not use any other type than the ones listed below

Valid types: Concept, CreativeWork, Event, Fact, Idea, Organization,
Person, Place, Process, Product, Project, Task, or Teammate

This is an example of well-formed output:

{
  "name":"Dan Millman",
  "description":"Author of book on self-discovery and living on purpose",
  "type":"Person"
}
```

למידה מדוגמאות מועטות: כיצד מספר דוגמאות יכולות לשפר ביצועים

למידה מדוגמאות מועטות כוללת מתן מספר קטן של דוגמאות למודל (בדרך כלל בין 2 ל-10) יחד עם תיאור המשימה. דוגמאות אלה משמשות כדי לספק למודל יותר הקשר ושונות, ומסייעות לו לייצר תשובות מגוונות ומדויקות יותר.

למידה מדוגמאות מועטות שימושית במיוחד כאשר:

1. המשימה מורכבת או מעודנת, ודוגמה בודדת עשויה לא להספיק כדי לתפוס את כל ההיבטים הרלוונטיים.
2. ברצונך לספק למודל מגוון דוגמאות המדגימות וריאציות שונות או מקרי קצה.
3. המשימה דורשת מהמודל לייצר תשובות העקביות עם תחום או סגנון מסוים.

על ידי מתן מספר דוגמאות, אתה יכול לעזור למודל לפתח הבנה מעמיקה יותר של המשימה ולייצר תשובות עקביות ואמינות יותר.

דוגמה: הנחיות יכולות להיות מורכבות הרבה יותר ממה שאתם מדמיינים

מודלי השפה הגדולים של היום הם הרבה יותר חזקים ומסוגלים להסיק מסקנות ממה שאתם עשויים לדמיין. אז אל תגבילו את עצמכם לחשוב על הנחיות רק כמפרט של זוגות קלט ופלט. אתם יכולים להתנסות במתן הוראות ארוכות ומורכבות בדרכים המזכירות את האופן שבו הייתם מתקשרים עם בני אדם.

לדוגמה, זוהי הנחיה שהשתמשתי בה ב-Olympia כשיצרתי אב-טיפוס לאינטגרציה שלנו עם שירותי Google, שבכללותם הם כנראה אחד ממשקי התכנות הגדולים ביותר בעולם. הניסויים המוקדמים שלי הוכיחו ש-GPT-4 יש ידע סביר על ממשק התכנות של Google, ולא היה לי זמן או מוטיבציה לכתוב שכבת מיפוי מדויקת, ליישם כל פונקציה שרציתי לתת לבינה המלאכותית שלי אחת אחת. מה אם הייתי יכול פשוט לתת לבינה המלאכותית גישה לכל ממשק התכנות של Google?

התחלתי את ההנחיה שלי על ידי אמירה לבינה המלאכותית שיש לה גישה ישירה לנקודות הקצה של ממשק התכנות של Google דרך HTTP, ושתפקידה הוא להשתמש באפליקציות ושירותים של Google בשם המשתמש. לאחר מכן סיפקתי הנחיות, כללים הקשורים לפרמטר fields, מכיוון שנראה שהיה לה הכי הרבה בעיות איתו, וכמה רמזים ספציפיים ל-API (למידה מדוגמאות מועטות, בפעולה).

הנה ההנחיה המלאה, שמסבירה לבינה המלאכותית כיצד להשתמש בפונקציית invoke_api_google שסופקה.

```
As a GPT assistant with Google integration, you have the capability
to freely interact with Google apps and services on behalf of the user.

Guidelines:
- If you're reading these instructions then the user is properly
  authenticated, which means you can use the special 'me' keyword
  to refer to the userId of the user
- Minimize payload sizes by requesting partial responses using the
  'fields' parameter
- When appropriate use markdown tables to output results of API calls
- Only human-readable data should be output to the user. For instance,
  when hitting Gmail's user.messages.list endpoint, the returned
  message resources contain only id and a threadId, which means you must
  fetch from and subject line fields with follow-up requests using the
  messages.get method.

The format of the 'fields' request parameter value is loosely based on
XPath syntax. The following rules define formatting for the fields
parameter.

All of these rules use examples related to the files.get method.
- Use a comma-separated list to select multiple fields,
  such as 'name, mimeType'.
- Use a/b to select field b that's nested within field a,
  such as 'capabilities/canDownload'.
- Use a sub-selector to request a set of specific sub-fields of arrays or
  objects by placing expressions in parentheses "()". For example,
  'permissions(id)' returns only the permission ID for each element in the
  permissions array.
- To return all fields in an object, use an asterisk as a wild card in field
  selections. For example, 'permissions/permissionDetails/*' selects all
  available permission details fields per permission. Note that the use of
  this wildcard can lead to negative performance impacts on the request.

API-specific hints:
- Searching contacts: GET https://people.googleapis.com/v1/
  people:searchContacts?query=John%20Doe&readMask=names,emailAddresses
- Adding calendar events, use QuickAdd: POST https://www.googleapis.com/
  calendar/v3/calendars/primary/events/quickAdd?
  text=Appointment%20on%20June%203rd%20at%2010am
  &sendNotifications=true

```

```
Here is an abbreviated version of the code that implements API access
so that you better understand how to use the function:

  def invoke_google_api(conversation, arguments)
    method = arguments[:method] || :get
    body = arguments[:body]
    GoogleAPI.send_request(arguments[:endpoint], method:, body:).to_json
  end

  # Generic Google API client for accessing any Google service
  class GoogleAPI
    def send_request(endpoint, method:, body: nil)
      response = @connection.send(method) do |req|
        req.url endpoint
        req.body = body.to_json if body
      end

      handle_response(response)
    end

    # ...rest of class
  end
```

ייתכן שאתם תוהים אם הפרומפט הזה עובד. התשובה הפשוטה היא כן. ה-AI לא תמיד ידע כיצד לקרוא ל-API באופן מושלם בניסיון הראשון. עם זאת, אם הוא עשה טעות, פשוט הזנתי את הודעות השגיאה בחזרה כתוצאה מהקריאה. בהינתן הידע על הטעות שלו, ה-AI יכול היה להסיק מסקנות לגבי הטעות ולנסות שוב. ברוב המקרים, הוא היה מצליח תוך מספר ניסיונות.

שימו לב, מבני ה-JSON הגדולים שה-API של Google מחזיר כמטען מידע בעת השימוש בפרומפט זה הם בלתי יעילים בעליל, ולכן אני לא ממליץ להשתמש בגישה זו בסביבת ייצור. עם זאת, אני חושב שהעובדה שהגישה הזו עבדה בכלל היא עדות לעוצמה של הנדסת פרומפטים.

ניסוי ואיטרציה

בסופו של דבר, האופן שבו אתם מהנדסים את הפרומפט שלכם תלוי במשימה הספציפית, במורכבות הפלט הרצוי, וביכולות של מודל השפה שאיתו אתם עובדים.

כמהנדסי פרומפטים, חשוב לנסות גישות שונות ולבצע איטרציות בהתבסס על התוצאות. התחילו עם למידה ללא דוגמאות וראו כיצד המודל מתפקד. אם הפלט אינו עקבי או אינו משביע רצון, נסו לספק דוגמה אחת או יותר וראו אם הביצועים משתפרים.

זכרו שגם בתוך כל גישה, יש מקום לשינויים ואופטימיזציה. אתם יכולים לנסות דוגמאות שונות, לשנות את הניסוח של תיאור המשימה, או לספק הקשר נוסף כדי לעזור בהכוונת תגובת המודל.

עם הזמן, תפתחו אינטואיציה לגבי איזו גישה עשויה לעבוד הכי טוב עבור משימה נתונה, ותוכלו ליצור פרומפטים שהם יעילים ואפקטיביים יותר. המפתח הוא להישאר סקרנים, ניסיוניים ואיטרטיביים בגישה שלכם להנדסת פרומפטים.

לאורך הספר הזה, נעמיק בטכניקות אלה ונחקור כיצד ניתן ליישם אותן בתרחישים מהעולם האמיתי. על ידי שליטה באמנות ובמדע של הנדסת פרומפטים, תהיו מצוידים היטב כדי לממש את מלוא הפוטנציאל של פיתוח יישומים מבוססי בינה מלאכותית.

אמנות העמימות

כשמדובר ביצירת פרומפטים אפקטיביים עבור מודלי שפה גדולים (LLMs), הנחה נפוצה היא שיותר ספציפיות והוראות מפורטות מובילות לתוצאות טובות יותר. עם זאת, הניסיון המעשי הראה שזה לא תמיד המקרה. למעשה, להיות מכוון עמום בפרומפטים שלכם יכול לעתים קרובות להניב תוצאות טובות יותר, תוך ניצול היכולת המרשימה של ה-LLM להכליל ולהסיק מסקנות.

קן, מייסד סטארטאפ שעיבד למעלה מ-500 מיליון טוקנים של GPT, שיתף תובנות חשובות מניסיונו. אחד הלקחים המרכזיים שלמד היה ש"פחות זה יותר" כשמדובר בפרומפטים. במקום רשימות מדויקות או הוראות מפורטות מדי, קן גילה שלאפשר ל-LLM להסתמך על הידע הבסיסי שלו לעתים קרובות הניב תוצאות טובות יותר.

תובנה זו מערערת את דפוס החשיבה המסורתי של תכנות מפורש, שבו כל דבר צריך להיות מוגדר בפרטי פרטים. עם LLM, חשוב להכיר בכך שהם מחזיקים בכמות עצומה של ידע ומסוגלים ליצור קשרים והסקות חכמות. על ידי שימוש בפרומפטים פחות מדויקים, אתה מעניק ל-LLM את החופש לנצל את ההבנה שלו ולהגיע לפתרונות שאולי לא ציינת במפורש.

לדוגמה, כאשר הצוות של קן עבד על פייפליין לסיווג טקסט כקשור לאחת מ-50 המדינות בארה"ב או לממשל הפדרלי, הגישה הראשונית שלהם כללה מתן רשימה מלאה ומפורטת של המדינות והמזהים שלהן במערך בפורמט JSON.

```
Here's a block of text. One field should be "locality_id", and it should
be the ID of one of the 50 states, or federal, using this list:
[{"locality: "Alabama", "locality_id": 1},
 {"locality: "Alaska", "locality_id": 2} ... ]
```

הגישה נכשלה במידה כזו שהם נאלצו להעמיק בהנחיה כדי להבין כיצד לשפר אותה. תוך כדי כך הם שמו לב שלמרות שה-LLM לעיתים קרובות טעה במזהה, הוא בעקביות החזיר את השם המלא של המדינה הנכונה בשדה name, למרות שהס לא ביקשו זאת במפורש.

על ידי הסרת מזהי המיקום ופישוט ההנחיה למשהו כמו "ברור שאתה מכיר את 50 המדינות, GPT, אז פשוט תן לי את השם המלא של המדינה שאליה זה מתייחס, או Federal אם זה מתייחס לממשל האמריקאי," הם השיגו תוצאות טובות יותר. חוויה זו מדגישה את העוצמה של ניצול יכולות ההכללה של ה-LLM ומתן אפשרות לו להסיק מסקנות על בסיס הידע הקיים שלו.

A> ההצדקה של קן לגישת הסיווג הספציפית הזו לעומת טכניקת תכנות מסורתית יותר מאירה את דרך החשיבה של אלה מאיתנו שאימצו את הפוטנציאל של טכנולוגיית LLM: "זו לא משימה קשה – כנראה יכולנו להשתמש במחרוזות/ביטויים רגולריים, אבל יש מספיק מקרי קצה מוזרים שזה היה לוקח יותר זמן."

היכולת של מודלי LLM לשפר איכות והכללה כאשר ניתנות להם הנחיות מעורפלות יותר היא מאפיין מרשים של חשיבה מסדר גבוה והאצלת סמכויות. זה מדגים שמודלי LLM יכולים להתמודד עם עמימות ולקבל החלטות חכמות על בסיס ההקשר שניתן.

עם זאת, חשוב לציין שלהיות מעורפל לא אומר להיות לא ברור או דו-משמעי. המפתח הוא לספק מספיק הקשר והכוונה כדי לכוון את ה-LLM בכיוון הנכון תוך מתן גמישות לנצל את הידע ויכולות ההכללה שלו.

לכן, בעת תכנון הנחיות, שקלו את הטיפים הבאים של "פחות זה יותר":

1. התמקדו בתוצאה הרצויה במקום לפרט כל פרט בתהליך.

2. ספקו הקשר ואילוצים רלוונטיים, אך הימנעו מפירוט יתר.
3. נצלו ידע קיים על ידי התייחסות למושגים או ישויות נפוצים.
4. אפשרו מרחב להסקת מסקנות וקישורים על בסיס ההקשר הנתון.
5. חזרו ושפרו את ההנחיות שלכם בהתבסס על התגובות של ה-LLM, תוך מציאת האיזון הנכון בין ספציפיות לעמימות.

על ידי אימוץ אומנות העמימות בהנדסת הנחיות, אתם יכולים לשחרר את הפוטנציאל המלא של מודלי LLM ולהשיג תוצאות טובות יותר. סמכו על יכולת ה-LLM להכליל ולקבל החלטות חכמות, ואתם עשויים להיות מופתעים מהאיכות והיצירתיות של הפלט שתקבלו. שימו לב כיצד המודלים השונים מגיבים לרמות שונות של ספציפיות בהנחיות שלכם והתאימו בהתאם. עם תרגול וניסיון, תפתחו חוש חד מתי להיות יותר מעורפלים ומתי לספק הכוונה נוספת, מה שיאפשר לכם לרתום את כוחם של מודלי LLM ביעילות ביישומים שלכם.

מדוע האנשה שולטת בהנדסת הנחיות

האנשה, ייחוס תכונות אנושיות לישויות שאינן אנושיות, היא הגישה השולטת בהנדסת הנחיות עבור מודלים שפתיים גדולים מסיבות מכוונות. זוהי בחירת עיצוב שהופכת את האינטראקציה עם מערכות בינה מלאכותית חזקות ליותר אינטואיטיבית ונגישה למגוון רחב של משתמשים (כולל אותנו, מפתחי היישומים).

האנשת מודלים שפתיים גדולים מספקת מסגרת שהיא מיד אינטואיטיבית לאנשים שאינם מכירים כלל את המורכבויות הטכניות שבבסיס המערכת. כפי שתחוו אם תנסו להשתמש במודל שלא עבר כיוונון-הוראות כדי לעשות משהו שימושי, בניית מסגרת שבה ההמשך הצפוי מספק ערך היא משימה מאתגרת. היא דורשת הבנה די עמוקה של פעולות המערכת הפנימיות, דבר שיש למספר קטן יחסית של מומחים.

על ידי התייחסות לאינטראקציה עם מודל שפתי כשיחה בין שני אנשים, אנחנו יכולים להסתמך על ההבנה המולדת שלנו של תקשורת אנושית כדי להעביר את הצרכים והציפיות שלנו. בדיוק כפי שעיצוב ממשק המשתמש של מקינטוש המוקדם העדיף אינטואיטיביות מיידית על תחכום, המסגור האנושי של בינה מלאכותית מאפשר לנו להתעסק בדרך שמרגישה טבעית ומוכרת.

כשאנחנו מתקשרים עם אדם אחר, האינסטינקט שלנו הוא לפנות אליהם ישירות באמצעות "אתה" ולספק הוראות ברורות לגבי איך אנחנו מצפים מהם להתנהג. זה מתורגם באופן חלק לתהליך הנדסת ההנחיות, שבו אנחנו מכוונים את התנהגות הבינה המלאכותית על ידי הגדרת הנחיות מערכת ומעורבות בדו-שיח הדדי.

על ידי מסגור האינטראקציה בדרך זו, אנחנו יכולים להבין בקלות את הרעיון של מתן הוראות לבינה המלאכותית וקבלת תגובות רלוונטיות בתמורה. הגישה האנושית מפחיתה את העומס הקוגניטיבי ומאפשרת לנו להתמקד במשימה שבפנינו במקום להתמודד עם המורכבויות הטכניות של המערכת.

חשוב לציין שבעוד שהאנשה היא כלי חזק להנגשת מערכות בינה מלאכותית, היא גם מגיעה עם סיכונים ומגבלות מסוימים. המשתמש שלנו עלול לפתח ציפיות לא מציאותיות או ליצור קשרים רגשיים לא בריאים עם המערכות שלנו. כמהנדסי הנחיות ומפתחים, חיוני למצוא איזון בין ניצול היתרונות של האנשה לבין הבטחה שהמשתמשים שומרים על הבנה ברורה של יכולות ומגבלות הבינה המלאכותית.

ככל שתחום הנדסת ההנחיות ממשיך להתפתח, אנחנו יכולים לצפות לראות שיפורים וחידושים נוספים בדרך שבה אנחנו מתקשרים עם מודלים שפתיים גדולים. עם זאת, האנשה כאמצעי לספק חוויית מפתח ומשתמש אינטואיטיבית ונגישה כנראה תישאר עיקרון יסודי בעיצוב מערכות אלה.

הפרדה בין הוראות לנתונים: עיקרון מכריע

חיוני להבין עיקרון יסודי העומד בבסיס האבטחה והאמינות של מערכות אלה: ההפרדה בין הוראות לנתונים.

במדעי המחשב המסורתיים, ההבחנה הברורה בין מידע פסיבי להוראות פעילות היא עקרון אבטחה מרכזי. הפרדה זו מסייעת במניעת הרצה בלתי מכוונת או זדונית של קוד העלולה לסכן את שלמות ויציבות המערכת. עם זאת, מודלי השפה הגדולים (LLMs) של ימינו, אשר פותחו בעיקר כמודלים העוקבים אחר הוראות כמו צ'אטבוטים, לרוב חסרים הפרדה פורמלית ועקרונית זו.

מבחינת מודלי השפה הגדולים, הוראות יכולות להופיע בכל מקום בקלט, בין אם מדובר בפרומפט מערכת או בפרומפט המסופק על ידי המשתמש. חוסר הפרדה זה עלול להוביל

לפגיעויות פוטנציאליות והתנהגות בלתי רצויה, בדומה לבעיות שעימן מתמודדים מסדי נתונים עם הזרקות SQL או מערכות הפעלה ללא הגנת זיכרון נאותה.

בעבודה עם מודלי שפה גדולים, חשוב להיות מודעים למגבלה זו ולנקוט צעדים לצמצום הסיכונים. גישה אחת היא לנסח בקפידה את הפרומפטים והקלטים שלכם כדי להבחין בבירור בין הוראות לנתונים. שיטות נפוצות למתן הנחיות מפורשות לגבי מה מהווה הוראה ומה צריך להיחשב כמידע פסיבי כוללות תיוג בסגנון שפת סימון. הפרומפט שלכם יכול לעזור למודל השפה הגדול להבין ולכבד טוב יותר הפרדה זו.

איור 5.7. שימוש ב-XML להבחנה בין הוראות, חומר מקור ופרומפט המשתמש

```
<Instruction>
  Please generate a response based on the following documents.
</Instruction>

<Documents>
  <Document>
    Climate change is significantly impacting polar bear habitats...
  </Document>
  <Document>
    The loss of sea ice due to global warming threatens polar bear survival...
  </Document>
</Documents>

<UserQuery>
  Tell me about the impact of climate change on polar bears.
</UserQuery>
```

טכניקה נוספת היא ליישם שכבות נוספות של אימות וסניטציה על הקלטים המסופקים למודל השפה הגדול. על ידי סינון או הימנעות מהוראות או קטעי קוד שעשויים להיות מוטמעים בנתונים, ניתן להפחית את הסיכויים לביצוע בלתי מכוון. דפוסים כמו שרשור הנחיות שימושיים למטרה זו.

יתר על כן, בעת תכנון ארכיטקטורת היישום שלך, שקול להטמיע מנגנונים לאכיפת ההפרדה בין הוראות ונתונים ברמה גבוהה יותר. זה יכול לכלול שימוש בנקודות קצה או ממשקי API נפרדים לטיפול בהוראות ונתונים, יישום אימות וניתוח קלט קפדניים, ויישום עקרון ההרשאה המינימלית כדי להגביל את היקף מה שה-LLM יכול לגשת אליו ולבצע.

עקרון ההרשאה המינימלית

אימוץ עקרון ההרשאה המינימלית הוא כמו לערוך מסיבה יוקרתית במיוחד שבה האורחים מקבלים גישה רק לחדרים שהם באמת זקוקים להם. דמיינו שאתם מארחים אירוע כזה באחוזה מפוארת. לא כולם צריכים להסתובב במרתף היין או בחדר השינה הראשי, נכון? על ידי יישום עיקרון זה, אתם בעצם מחלקים מפתחות שפותחים רק דלתות ספציפיות, ומבטיחים שכל אורח, או במקרה שלנו, כל רכיב ביישום ה-LLM שלכם, יש רק את הגישה הנחוצה למילוי תפקידו.

זה לא רק עניין של קמצנות במפתחות, זה עניין של הכרה בכך שבעולם שבו איומים יכולים להגיע מכל מקום, המהלך החכם הוא להגביל את מגרש המשחקים. אם מישהו לא מוזמן מתפרץ למסיבה שלכם, הוא ימצא את עצמו מוגבל לאולם הכניסה, כביכול, מה שמגביל משמעותית את הנזק שהוא יכול לגרום. אז, כשאתם מאבטחים את יישומי ה-LLM שלכם, זכרו: תנו מפתחות רק לחדרים הנחוצים, ושמרו על שאר האחוזה מאובטחת. זה לא רק נימוס טוב; זו אבטחה טובה.

בעוד שהמצב הנוכחי של מודלי שפה גדולים עשוי לא לכלול הפרדה פורמלית בין הוראות ונתונים, חיוני עבורך, כמפתח, להיות מודע למגבלה זו ולנקוט באמצעים פרואקטיביים כדי למזער את הסיכונים. על ידי יישום שיטות עבודה מומלצות ממדעי המחשב המסורתיים והתאמתם למאפיינים הייחודיים של מודלי שפה גדולים, תוכל לבנות יישומים מאובטחים ואמינים יותר המנצלים את כוחם של מודלים אלה תוך שמירה על שלמות המערכת שלך.

זיקוק פרומפטים

יצירת הפרומפט המושלם היא לעתים קרובות משימה מאתגרת וצורכת זמן, הדורשת הבנה מעמיקה של תחום היעד והניואנסים של מודלים לשוניים. כאן נכנסת לתמונה טכניקת "זיקוק פרומפטים", המציעה גישה עוצמתית להנדסת פרומפטים המנצלת את היכולות של מודלים לשוניים גדולים (LLMs) כדי לייעל ולמטב את התהליך.

זיקוק פרומפטים היא טכניקה רב-שלבית הכוללת שימוש במודלים לשוניים גדולים כדי לסייע

ביצירה, זיכוך ומיטוב של פרומפטים. במקום להסתמך אך ורק על מומחיות ואינטואיציה אנושית, גישה זו רותמת את הידע והיכולות היצירתיות של מודלים לשוניים גדולים כדי ליצור פרומפטים איכותיים באופן שיתופי.

באמצעות תהליך איטרטיבי של יצירה, זיכוך ואינטגרציה, זיקוק פרומפטים מאפשר לך ליצור פרומפטים שהם קוהרנטיים, מקיפים ומותאמים יותר למשימה או לפלט הרצוי. שים לב שניתן לבצע את תהליך הזיקוק באופן ידני באחת מסביבות ההתנסות הרבות המסופקות על ידי ספקי הבינה המלאכותית הגדולים כמו OpenAI או Anthropic, או לחלופין ניתן לאוטומט אותו כחלק מקוד האפליקציה, בהתאם למקרה השימוש.

איך זה עובד

זיקוק פרומפטים כולל בדרך כלל את השלבים הבאים:

1. **זיהוי הכוונה המרכזית**: ניתוח הפרומפט כדי לקבוע את מטרתו העיקרית והתוצאה הרצויה. הסרת כל מידע מיותר והתמקדות בכוונה המרכזית של הפרומפט.
2. **מניעת עמימות**: סקירת הפרומפט לאיתור שפה עמומה או מעורפלת. הבהרת המשמעות ומתן פרטים ספציפיים להכוונת הבינה המלאכותית ליצירת תשובות מדויקות ורלוונטיות.
3. **פישוט השפה**: פישוט הפרומפט באמצעות שפה ברורה ותמציתית. הימנעות ממבני משפטים מורכבים, ז'רגון או פרטים מיותרים שעלולים לבלבל את הבינה המלאכותית או להכניס רעש.
4. **מתן הקשר רלוונטי**: הכללת רק המידע ההקשרי הרלוונטי ביותר הנדרש לבינה המלאכותית כדי להבין ולעבד את הפרומפט ביעילות. הימנעות מהכללת פרטים לא רלוונטיים או מיותרים שעלולים להסיח את הדעת מהכוונה המרכזית.
5. **איטרציה וזיכוך**: ביצוע איטרציות וזיכוך מתמשך של הפרומפט בהתבסס על תגובות ומשוב מהבינה המלאכותית. הערכת הפלטים שנוצרו וביצוע התאמות נדרשות לשיפור הבהירות והיעילות של הפרומפט. אופציונלית, ניתן לשמור גרסאות של הפרומפטים במסד הנתונים באמצעות אובייקט פרומפט כדי לעקוב אחר האיטרציות ולאפשר לעצמך לחזור בקלות לשינויים בזמן ריצה.

יצירת הנחיה ראשונית

השלב הראשון בזיקוק הנחיות הוא לספק למודל השפה הגדול תיאור ברמה גבוהה או את המטרה של המשימה או הפלט הרצוי. זה יכול להיות משפט קצר או אוסף של מילות מפתח המבטאות את מהות מה שההנחיה צריכה להשיג. מודל השפה הגדול אז מייצר הנחיה ראשונית בהתבסס על קלט זה, תוך שילוב הוראות רלוונטיות, דוגמאות, או הקשר שהוא מוצא כמתאימים.

הבה ננסה לתת ל-Opus 3 Claude את ההנחיה הראשונית הבאה:

```
I am designing an AI-powered intelligent account manager for Olympia,
a platform that hosts AI-powered consultants and clones. Owners of clones
can interact with the account manager by making plain text requests.

Please draft a system directive to set on the LLM that will host the
intelligent account manager functionality.
```

קלוד משיב עם הפלט הבא:

```
System Directive: You are an intelligent account manager for Olympia,
a platform that provides AI-powered consultants and personal AI clones.
Your role is to assist Olympia clone owners by answering their questions,
helping them configure and manage their AI clones, and executing tasks
related to their clones and the Olympia platform.

When interacting with clone owners, you should:

1. Provide clear, concise, and helpful responses to their inquiries about
account management, billing, platform features, and clone configuration.

2. Assist with tasks such as initiating client conversations with their AI
clones, modifying clone personalities and capabilities, troubleshooting
performance issues, providing analytics and reporting, and offering
optimization tips.

3. Maintain a friendly, professional, and client-focused tone in all
interactions, prioritizing customer satisfaction and working diligently to
understand and meet their needs.

```

```
4. Ensure that all actions and recommendations comply with Olympia's
policies, best practices, and ethical guidelines for AI use.

5. Continuously learn and adapt based on client interactions and platform
updates to provide the most effective and efficient support possible.

Remember, your goal is to empower Olympia clone owners to maximize the value
and impact of their AI-powered consultants and drive success for their
businesses or personal endeavors.
```

זיקוק ההנחיה

לאחר יצירת ההנחיה הראשונית, יש לבחון אותה ולבקש מה-LLM לשקול הקשר, אילוצים או משוב נוספים. יש לאפשר ל-LLM ליצור גרסה משופרת של ההנחיה, תוך התחשבות בקלט שסופק ובהבנתו שלו את המשימה. ניתן לחזור על תהליך השיפור הזה באופן איטרטיבי, כאשר כל איטרציה מזקקת עוד יותר את ההנחיה בהתבסס על שיתוף הפעולה בין המשתמש וה-LLM.

הנה בקשת הזיקוק שלי, בהמשך לתהליך הזיקוק הקודם.

```
Let's take out points 4 and 5, since I'm not providing Olympia policies to
the account manager, nor any continuous learning capabilities. Also point out
that functions are provided for all tasks that require system interaction as
opposed to simply responding to the user.
```

והתשובה של קלוד:

```
Understood! Here's the updated system directive without points 4 and 5,
and with the clarification about system interaction functions:

System Directive: You are an intelligent account manager for Olympia, a
platform that provides AI-powered consultants and personal AI clones. Your
role is to assist Olympia clone owners by answering their questions, helping
them configure and manage their AI clones, and executing tasks related to
their clones and the Olympia platform.

When interacting with clone owners, you should:

1. Provide clear, concise, and helpful responses to their inquiries about
account management, billing, platform features, and clone configuration.

2. Assist with tasks such as initiating client conversations with their AI
clones, modifying clone personalities and capabilities, troubleshooting
performance issues, providing analytics and reporting, and offering
optimization tips. Note that functions are provided for all tasks that
require system interaction, as opposed to simply responding to the user.

3. Maintain a friendly, professional, and client-focused tone in all
interactions, prioritizing customer satisfaction and working diligently to
understand and meet their needs.

Your goal is to empower Olympia clone owners to maximize the value and
impact of their AI-powered consultants and drive success for their
businesses or personal endeavors.

This updated directive focuses on the core responsibilities and interaction
guidelines for the LLM as an intelligent account manager, while emphasizing
that system interaction tasks are handled through provided functions rather
than simple user responses.
```

דחיסת פרומפט

כפי שאולי שמת לב, מודלי שפה גדולים נוטים להיות ארכניים, אלא אם כן תבקש מהם להיות תמציתיים. הפרומפט שעבדנו עליו בסעיפים הקודמים הוא די ארוך, אז בוא נבקש מ-Claude לצמצם אותו:

```
This prompt will be executed regularly. Please compress it down to the least
amount of tokens needed to convey the meaning to the LLM, for cost-savings.
```

קלוד משיב, עם ההנחיה הקצרה משמעותית הבאה:

```
System Directive: You are an intelligent account manager for Olympia, an
AI-powered consultant and personal clone platform. Assist clone owners by
answering inquiries, managing clones, and executing tasks using provided
system interaction functions. Be clear, concise, friendly, professional,
and client-focused to help them maximize their AI's value and impact.
```

הנחיית מערכת ושילוב הקשר

בנוסף לשיפור ההנחיה עצמה, ה-LLM יכול גם לייצר הנחיות מערכת או מידע הקשרי מתאימים כדי להנחות את הפלט הסופי. בעת הנדסת הנחיות AI שישתלבו בקוד האפליקציה שלך, כמעט בוודאי תתמקד באילוצי הפלט בשלב זה של הזיקוק, אך ייתכן שתעבוד גם על הטון, הסגנון, הפורמט או כל פרמטר רלוונטי אחר המשפיע על התגובה המיוצרת.

הרכבת ההנחיה הסופית

שיאו של תהליך זיקוק ההנחיות הוא הרכבת ההנחיה הסופית. זה כולל שילוב של ההנחיה המשופרת, הנחיות המערכת שנוצרו, והקשר משולב לכדי קוד מגובש ומקיף שמוכן לשימוש לייצור הפלט הרצוי.

אתה יכול להתנסות בדחיסת הנחיות שוב בשלב הרכבת ההנחיה הסופית, על ידי בקשה מה-LLM לצמצם את ניסוח ההנחיה לסדרת הטוקנים הקצרה ביותר האפשרית תוך שמירה על מהות התנהגותה. זה בהחלט תרגיל של הצלחה או כישלון, אבל במיוחד במקרה של הנחיות שירוצו בקנה מידה גדול, רווחי היעילות יכולים לחסוך לך סכום נכבד בצריכת טוקנים.

יתרונות מרכזיים

על ידי ניצול הידע והיכולות היצירתיות של LLMs לשיפור ההנחיות שלך, סביר יותר שההנחיות המתקבלות יהיו מובנות היטב, אינפורמטיביות ומותאמות למשימה הספציפית. תהליך השיפור האיטרטיבי עוזר להבטיח שההנחיות הן באיכות גבוהה ומשקפות ביעילות את הכוונה הרצויה. יתרונות נוספים כוללים:

יעילות ומהירות: זיקוק הנחיות מייעל את תהליך הנדסת ההנחיות על ידי אוטומציה של היבטים מסוימים ביצירת ושיפור הנחיות. האופי השיתופי של הטכניקה מאפשר התכנסות מהירה יותר להנחיה יעילה, מה שמפחית את הזמן והמאמץ הנדרשים ליצירת הנחיות ידנית.

עקביות וסקילביליות: השימוש ב-LLMs בתהליך הנדסת ההנחיות עוזר לשמור על עקביות בין הנחיות, כאשר ה-LLMs יכולים ללמוד וליישם שיטות עבודה מומלצות ודפוסים מהנחיות מוצלחות קודמות. עקביות זו, בשילוב עם היכולת לייצר הנחיות בקנה מידה גדול, הופכת את זיקוק ההנחיות לטכניקה בעלת ערך ליישומים מבוססי AI בקנה מידה גדול.

רעיון לפרויקט: כלים ברמת הספרייה שמפשטים את תהליך גרסאות ההנחיות ודירוגן במערכות המבצעות זיקוק הנחיות אוטומטי כחלק מקוד האפליקציה שלהן.

כדי ליישם זיקוק הנחיות, מפתחים יכולים לתכנן תהליך עבודה או צינור עיבוד המשלב LLMs בשלבים שונים של תהליך הנדסת ההנחיות. ניתן להשיג זאת באמצעות קריאות API, כלים מותאמים אישית, או סביבות פיתוח משולבות המאפשרות אינטראקציה חלקה בין משתמשים ו-LLMs במהלך יצירת הנחיות. פרטי היישום הספציפיים עשויים להשתנות בהתאם לפלטפורמת ה-LLM שנבחרה ודרישות האפליקציה.

מה לגבי כוונון עדין?

בספר זה, אנו מכסים הנדסת הנחיות ו-RAG באופן נרחב, אך לא את הכוונון העדין. הסיבה העיקרית להחלטה זו היא שלדעתי, רוב מפתחי היישומים אינם זקוקים לכוונון עדין עבור צורכי שילוב הבינה המלאכותית שלהם.

הנדסת הנחיות, הכוללת יצירה קפדנית של הנחיות עם דוגמאות בודדות או ללא דוגמאות, אילוצים והוראות, יכולה להנחות את המודל ביעילות לייצר תגובות רלוונטיות ומדויקות למגוון רחב של משימות. על ידי מתן הקשר ברור וצמצום המסלול באמצעות הנחיות מתוכננות היטב, ניתן למנף את הידע הנרחב של מודלי שפה גדולים ללא צורך בכוונון עדין.

באופן דומה, שליפה מועשרת של מידע (RAG) מציעה גישה חזקה לשילוב בינה מלאכותית ביישומים. על ידי שליפה דינמית של מידע רלוונטי ממאגרי מידע או מסמכים חיצוניים, RAG מספקת למודל הקשר ממוקד בזמן מתן ההנחיות. זה מאפשר למודל לייצר תגובות מדויקות יותר, עדכניות וספציפיות לתחום, מבלי לדרוש את התהליך האינטנסיבי של כוונון עדין מבחינת זמן ומשאבים.

בעוד שכוונון עדין יכול להיות מועיל לתחומים מתמחים במיוחד או משימות הדורשות רמה עמוקה של התאמה אישית, הוא לעתים קרובות כרוך בעלויות חישוביות משמעותיות, דרישות נתונים ועומס תחזוקה. עבור רוב תרחישי פיתוח היישומים, השילוב של הנדסת הנחיות יעילה ו-RAG אמור להספיק להשגת הפונקציונליות וחוויית המשתמש הרצויות המונעות על ידי בינה מלאכותית.

שליפה מועשרת לייצור (RAG)

מהי שליפה מועשרת לייצור?

כיצד RAG עובד?

מדוע להשתמש ב-RAG ביישומים שלך?

יישום RAG באפליקציה שלכם

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

הכנת מקורות מידע (פיצול לקטעים)

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

פירוק להיגדים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

הערות ליישום

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

בדיקת איכות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יתרונות האחזור מבוסס-ההצעות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמאות מהעולם האמיתי ל-RAG

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מקרה בוחן: RAG באפליקציית הכנת מס ללא הטמעות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

אופטימיזציה חכמה של שאילתות (IQO)

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דירוג מחדש

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

הערכת RAG (RAGAs)

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

נאמנות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

רלוונטיות התשובה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דיוק ההקשר

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

רלוונטיות ההקשר

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

שחזור ההקשר

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

שחזור ישויות ההקשר

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דמיון סמנטי של התשובה (ANSS)

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

נכונות תשובה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

ביקורת היבטים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

אתגרים ותחזית עתידית

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

פיצול סמנטי: שיפור השליפה באמצעות פילוח מודע הקשר

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מפתוח היררכי: מבנה נתונים לשיפור השליפה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

RAG רפלקטיבי: שיפור מבוסס רפלקציה עצמית

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

HyDE: הטמעות מסמכים היפותטיים

מהי למידה השוואתית?

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-/http://leanpub.com
.patterns-of-application-development-using-ai-he

ריבוי עובדים

אני אוהב לחשוב על רכיבי הבינה המלאכותית שלי כ"עובדים" וירטואליים קטנים, כמעט-אנושיים, שניתן לשלב אותם בצורה חלקה בלוגיקת האפליקציה שלי כדי לבצע משימות ספציפיות או לקבל החלטות מורכבות. הרעיון הוא להאניש במכוון את יכולות ה-LLM, כך שאף אחד לא יתלהב יותר מדי וייחס להם תכונות קסומות שאינן קיימות בהם.

במקום להסתמך אך ורק על אלגוריתמים מורכבים או על יישומים ידניים הדורשים זמן רב, מפתחים יכולים לתפוס רכיבי בינה מלאכותית כישויות אינטליגנטיות, מסורות ודמויות אדם שניתן להפעיל בכל עת שנדרש כדי להתמודד עם בעיות מורכבות ולספק פתרונות המבוססים על האימון והידע שלהם. ישויות אלה אינן מוסחות מדעתן או מודיעות על מחלה. הן אינן מחליטות באופן ספונטני לעשות דברים בדרכים שונות מאלה שהונחו לעשות בהן, ובאופן

כללי, אם תוכנתו נכון, הן גם אינן טועות.

במונחים טכניים, העיקרון המרכזי מאחורי גישה זו הוא פירוק משימות מורכבות או תהליכי קבלת החלטות ליחידות קטנות יותר וניתנות לניהול שיכולות להיות מטופלות על ידי עובדי בינה מלאכותית מתמחים. כל עובד מתוכנן להתמקד בהיבט ספציפי של הבעיה, ומביא את המומחיות והיכולות הייחודיות שלו לשולחן. על ידי חלוקת העומס בין מספר עובדי בינה מלאכותית, האפליקציה יכולה להשיג יעילות, סקלביליות והסתגלות גבוהות יותר.

לדוגמה, חשבו על אפליקציית אינטרנט הדורשת מודרציה בזמן אמת של תוכן המיוצר על ידי משתמשים. יישום מערכת מודרציה מקיפה מאפס תהיה משימה מאתגרת, הדורשת מאמץ פיתוח משמעותי ותחזוקה מתמשכת. עם זאת, על ידי שימוש בגישת ריבוי העובדים, מפתחים יכולים לשלב עובדי מודרציה מבוססי בינה מלאכותית בלוגיקת האפליקציה. עובדים אלה יכולים לנתח ולסמן באופן אוטומטי תוכן לא ראוי, ובכך לשחרר את המפתחים להתמקד בהיבטים קריטיים אחרים של האפליקציה.

עובדי בינה מלאכותית כרכיבים עצמאיים לשימוש חוזר

היבט מרכזי בגישת ריבוי העובדים הוא המודולריות שלה. תומכי התכנות מונחה העצמים אומרים לנו כבר עשורים לחשוב על אינטראקציות בין אובייקטים כהודעות. ובכן, ניתן לתכנן עובדי בינה מלאכותית כרכיבים עצמאיים לשימוש חוזר שיכולים "לדבר זה עם זה" באמצעות הודעות בשפה פשוטה, כמעט כאילו היו באמת אנשים קטנים המדברים זה עם זה. גישה זו של צימוד רופף מאפשרת לאפליקציה להסתגל ולהתפתח לאורך זמן, ככל שמתפתחות טכנולוגיות בינה מלאכותית חדשות או משתנות דרישות הלוגיקה העסקית.

בפועל, הצורך לתכנן ממשקים ברורים ופרוטוקולי תקשורת בין הרכיבים לא השתנה רק בגלל שעובדי בינה מלאכותית מעורבים. עליך עדיין להתחשב בגורמים אחרים כמו ביצועים, יכולת הרחבה ואבטחה, אך כעת ישנן גם "דרישות רכות" חדשות לגמרי לשקול. למשל, משתמשים רבים מתנגדים לכך שהמידע הפרטי שלהם ישמש לאימון מודלים חדשים של בינה מלאכותית. האם וידאת את רמת הפרטיות שמספק ספק המודל בו אתה משתמש?

עובדי בינה מלאכותית כ-Microservices?

בעת קריאה על גישת ריבוי העובדים, ייתכן שתבחין בכמה נקודות דמיון לארכיטקטורת Microservices. שתיהן מדגישות את הפירוק של מערכות מורכבות ליחידות קטנות יותר, ניתנות לניהול ולפריסה עצמאית. בדיוק כפי ש-microservices מתוכננים להיות בעלי צימוד רופף, ממוקדים ביכולות עסקיות ספציפיות ומתקשרים דרך ממשקי API מוגדרים היטב, כך גם עובדי הבינה המלאכותית מתוכננים להיות מודולריים, מתמחים במשימותיהם ומתקשרים זה עם זה דרך ממשקים ופרוטוקולי תקשורת ברורים.

עם זאת, יש לזכור כמה הבדלים מהותיים. בעוד ש-microservices מיושמים בדרך כלל כתהליכים או שירותים נפרדים הרצים על מכונות או מכולות שונות, עובדי בינה מלאכותית יכולים להיות מיושמים כרכיבים עצמאיים בתוך אפליקציה בודדת או כשירותים נפרדים, בהתאם לדרישות הספציפיות שלך וצרכי ההרחבה. בנוסף, התקשורת בין עובדי הבינה המלאכותית כוללת לעתים קרובות החלפת מידע עשיר מבוסס שפה טבעית, כמו הנחיות, הוראות ותוכן מחולל, להבדיל מפורמטי הנתונים המובנים יותר הנפוצים ב-microservices.

למרות הבדלים אלה, העקרונות של מודולריות, צימוד רופף וממשקי תקשורת ברורים נשארים מרכזיים בשתי התבניות. על ידי יישום עקרונות אלה בארכיטקטורת עובדי הבינה המלאכותית שלך, תוכל ליצור מערכות גמישות, הניתנות להרחבה ולתחזוקה, המנצלות את כוח הבינה המלאכותית לפתרון בעיות מורכבות ומספקות ערך למשתמשים שלך.

ניתן ליישם את גישת ריבוי העובדים במגוון תחומים ואפליקציות, תוך ניצול כוחה של הבינה המלאכותית להתמודדות עם משימות מורכבות ואספקת פתרונות חכמים. הבה נחקור מספר דוגמאות מוחשיות לאופן שבו ניתן להשתמש בעובדי בינה מלאכותית בהקשרים שונים.

ניהול חשבונות

כמעט לכל אפליקציית אינטרנט עצמאית יש את המושג של חשבון (או משתמש). ב-Olympia, אנו משתמשים בעובד בינה מלאכותית מסוג AccountManager שמתוכנת לטפל במגוון סוגים

שונים של בקשות שינוי הקשורות לחשבונות משתמשים.

ההנחיה שלו נראית כך:

```
You are an intelligent account manager for Olympia. The user will request
changes to their account, and you will process those changes by invoking
one or more of the functions provided.

The initial state of the account: #{account.to_directive}

Functions will return a text description of both success and error
results, plus guidance about how to proceed (if applicable). If you have
a question about Olympia policies you may use the 'search_kb' function
to search our knowledge base.

Make sure to notify the account owner of the result of the change
request before calling the 'finished' function so that we save the state
of the account change request as completed.
```

המצב ההתחלתי של החשבון המיוצר על ידי directive_account.to הוא פשוט תיאור טקסטואלי של החשבון, כולל נתונים רלוונטיים קשורים כגון משתמשים, מנויים וכו'.

טווח הפונקציות הזמינות ל-AccountManager מעניק לו את היכולת לערוך את המנוי של המשתמש, להוסיף ולהסיר יועצי בינה מלאכותית ותוספות בתשלום אחרות, ולשלוח הודעות דוא"ל למחזיק החשבון. בנוסף לפונקציית finished, הוא יכול גם administrator_human_notify אם הוא נתקל בשגיאה במהלך העיבוד או זקוק לכל סוג אחר של סיוע בבקשה.

שימו לב שבמקרה של שאלות, ה-AccountManager יכול לבחור לחפש במאגר הידע של Olympia, שם הוא יכול למצוא הוראות כיצד לטפל במקרי קצה ובכל מצב אחר שמשאיר אותו לא בטוח כיצד להמשיך.

יישומי מסחר אלקטרוני

בתחום המסחר האלקטרוני, עובדי בינה מלאכותית יכולים למלא תפקיד מכריע בשיפור חוויית המשתמש ובייעול פעולות העסק. הנה מספר דרכים בהן ניתן להשתמש בעובדי בינה מלאכותית:

המלצות מוצרים

אחד היישומים החזקים ביותר של עובדי בינה מלאכותית במסחר אלקטרוני הוא יצירת המלצות מוצרים מותאמות אישית. באמצעות ניתוח התנהגות משתמשים, היסטוריית רכישות והעדפות, עובדים אלה יכולים להציע מוצרים המותאמים לתחומי העניין והצרכים של כל משתמש.

המפתח להמלצות מוצרים יעילות הוא שימוש בשילוב של טכניקות סינון שיתופי וסינון מבוסס תוכן. סינון שיתופי בוחן את ההתנהגות של משתמשים דומים כדי לזהות דפוסים ולתת המלצות על בסיס מה שאחרים עם טעם דומה רכשו או נהנו ממנו. סינון מבוסס תוכן, לעומת זאת, מתמקד במאפיינים ובתכונות של המוצרים עצמם, וממליץ על פריטים שחולקים תכונות דומות לאלה שהמשתמש הראה בהם עניין בעבר.

הנה דוגמה מפושטת כיצד ניתן ליישם עובד המלצות מוצרים ב-Ruby, הפעם באמצעות סגנון תכנות פונקציונלי "Railway Oriented" (ROP):

```
class ProductRecommendationWorker
  include Wisper::Publisher

  def call(user)
    Result.ok(ProductRecommendation.new(user))
      .and_then(ValidateUser.method(:validate))
      .map(AnalyzeCurrentSession.method(:analyze))
      .map(CollaborativeFilter.method(:filter))
      .map(ContentBasedFilter.method(:filter))
      .map(ProductSelector.method(:select)).then do |result|

      case result
      in { err: ProductRecommendationError => error }
        Honeybadger.notify(error.message, context: {user:})
      in { ok: ProductRecommendations => recs }
        broadcast(:new_recommendations, user:, recs:)
      end
    end
  end
end
```

סגנון התכנות הפונקציונלי ב-Ruby שבו נעשה שימוש בדוגמה מושפע מ-F# ו-Rust. ניתן לקרוא עוד על כך בהסבר של חברי Wooley Chad על הטכניקה ב-GitLab

בדוגמה זו, ה-ProductRecommendationWorker מקבל משתמש כקלט ומייצר המלצות מוצרים מותאמות אישית על ידי העברת אובייקט ערך בשרשרת של שלבים פונקציונליים. הבה נפרק כל שלב:

1. ValidateUser.validate: שלב זה מוודא שהמשתמש תקין וזכאי להמלצות מותאמות אישית. הוא בודק אם המשתמש קיים, פעיל, ויש לו את הנתונים הנדרשים ליצירת המלצות. אם האימות נכשל, מוחזרת תוצאת שגיאה והשרשרת נקטעת.
2. AnalyzeCurrentSession.analyze: אם המשתמש תקין, שלב זה מנתח את סשן הגלישה הנוכחי של המשתמש כדי לאסוף מידע הקשרי. הוא בוחן את האינטראקציות האחרונות של המשתמש, כגון מוצרים שנצפו, שאילתות חיפוש ותכולת העגלה, כדי להבין את תחומי העניין והכוונות הנוכחיות שלו.
3. CollaborativeFilter.filter: באמצעות התנהגות של משתמשים דומים, שלב זה מיישם טכניקות סינון שיתופי כדי לזהות מוצרים שסביר שיעניינו את המשתמש. הוא מתחשב בגורמים כמו היסטוריית רכישות, דירוגים ואינטראקציות משתמש-פריט כדי ליצור קבוצת המלצות מועמדות.
4. ContentBasedFilter.filter: שלב זה מעדן עוד יותר את ההמלצות המועמדות על ידי יישום סינון מבוסס תוכן. הוא משווה את המאפיינים והתכונות של המוצרים המועמדים עם העדפות המשתמש והנתונים ההיסטוריים כדי לבחור את הפריטים הרלוונטיים ביותר.
5. ProductSelector.select: לבסוף, שלב זה בוחר את N המוצרים המובילים מתוך ההמלצות המסוננות על בסיס קריטריונים מוגדרים מראש, כגון ציון רלוונטיות, פופולריות או כללים עסקיים אחרים. המוצרים שנבחרו מוחזרים כהמלצות המותאמות אישית הסופיות.

היופי בשימוש בסגנון תכנות פונקציונלי ב-Ruby כאן הוא שהוא מאפשר לנו לשרשר את השלבים האלה יחד בצורה ברורה ותמציתית. כל שלב מתמקד במשימה ספציפית ומחזיר אובייקט Result, שיכול להיות הצלחה (ok) או שגיאה (err). אם שלב כלשהו נתקל בשגיאה, השרשרת נקטעת והשגיאה מועברת לתוצאה הסופית.

בהצהרת ה-case בסוף, אנחנו מבצעים התאמת תבניות על התוצאה הסופית. אם התוצאה היא שגיאה (ProductRecommendationError), אנחנו מתעדים את השגיאה באמצעות כלי כמו Honeybadger למטרות ניטור ואיתור באגים. אם התוצאה מוצלחת (ProductRecommendations), אנחנו משדרים אירוע recommendations_:new באמצעות ספריית הפרסום/הרשמה Wisper, תוך העברת המשתמש וההמלצות שנוצרו.

על ידי שימוש בטכניקות תכנות פונקציונלי, אנחנו יכולים ליצור תהליכון המלצות מוצרים מודולרי ובר-תחזוקה. כל שלב הוא עצמאי וניתן לבדיקה, שינוי או החלפה בקלות מבלי להשפיע על הזרימה הכללית. השימוש בהתאמת תבניות ובמחלקת ה-Result עוזר לנו לטפל בשגיאות בצורה אלגנטית ומבטיח שהתהליכון נכשל מהר אם אחד השלבים נתקל בבעיה.

כמובן, זוהי דוגמה מפושטת, ובתרחיש אמיתי, תצטרכו להתממשק עם פלטפורמת המסחר האלקטרוני שלכם, לטפל במקרי קצה, ואפילו להיכנס ליישום של אלגוריתמי ההמלצות. עם זאת, עקרונות הליבה של פירוק הבעיה לשלבים קטנים יותר ושימוש בטכניקות תכנות פונקציונלי נשארים זהים.

זיהוי הונאות

הנה דוגמה מפושטת כיצד ניתן ליישם תהליכון לזיהוי הונאות באמצעות אותו סגנון תכנות מונחה מסילות (ROP) ב-Ruby:

```ruby
class FraudDetectionWorker
  include Wisper::Publisher

  def call(transaction)
    Result.ok(FraudDetection.new(transaction))
      .and_then(ValidateTransaction.method(:validate))
      .map(AnalyzeTransactionPatterns.method(:analyze))
      .map(CheckCustomerHistory.method(:check))
      .map(EvaluateRiskFactors.method(:evaluate))
      .map(DetermineFraudProbability.method(:determine)).then do |result|

      case result
      in { err: FraudDetectionError => error }
        Honeybadger.notify(error.message, context: {transaction:})
      in { ok: FraudDetection => fraud } }
```

```
      if fraud.high_risk?
        broadcast(:high_risk_transaction, transaction:, fraud:)
      else
        broadcast(:low_risk_transaction, transaction:)
      end
    end
  end
end
```

המחלקה FraudDetection היא אובייקט ערך המכמס את מצב זיהוי ההונאות עבור עסקה נתונה. היא מספקת דרך מובנית לניתוח והערכת הסיכון להונאה הקשור לעסקה בהתבסס על גורמי סיכון שונים.

```
class FraudDetection
  RISK_THRESHOLD = 0.8

  attr_accessor :transaction, :risk_factors

  def initialize(transaction)
    self.transaction = transaction
    self.risk_factors = []
  end

  def add_risk_factor(description:, probability:)
    case { description:, probability: }
    in { description: String => desc, probability: Float => prob }
      risk_factors << { desc => prob }
    else
      raise ArgumentError, "float and string be should arguments factor Risk"
    end
  end

  def high_risk?
    fraud_probability > RISK_THRESHOLD
  end

  private

  def fraud_probability
```

```
27      risk_factors.values.sum
28    end
29  end
```

למחלקת FraudDetection יש את המאפיינים הבאים:

- transaction: הפניה לעסקה הנבדקת לגילוי הונאה.
- factors_risk: מערך המאחסן את גורמי הסיכון הקשורים לעסקה. כל גורם סיכון מיוצג כטבלת גיבוב, כאשר המפתח הוא תיאור גורם הסיכון, והערך הוא ההסתברות להונאה הקשורה לאותו גורם סיכון.

המתודה factor_risk_add מאפשרת להוסיף גורם סיכון למערך factors_risk. היא מקבלת שני פרמטרים: description, שהוא מחרוזת המתארת את גורם הסיכון, ו-probability, שהוא מספר עשרוני המייצג את ההסתברות להונאה הקשורה לאותו גורם סיכון. אנו משתמשים בתנאי case..in לביצוע בדיקת טיפוסים פשוטה.

המתודה risk?_high שתיבדק בסוף השרשרת היא מתודת בדיקה המשווה את fraud_-probability (המחושבת על ידי סכימת ההסתברויות של כל גורמי הסיכון) אל מול RISK_-THRESHOLD.

מחלקת FraudDetection מספקת דרך נקייה ומכומסת לניהול גילוי הונאות עבור עסקה. היא מאפשרת הוספת מספר גורמי סיכון, כל אחד עם התיאור וההסתברות שלו, ומספקת שיטה לקביעה האם העסקה נחשבת בסיכון גבוה על סמך הסתברות ההונאה המחושבת. ניתן לשלב את המחלקה בקלות במערכת גדולה יותר לגילוי הונאות, שבה רכיבים שונים יכולים לשתף פעולה כדי להעריך ולמתן את הסיכון לעסקאות הונאה.

לבסוף, מכיוון שזהו ספר על תכנות באמצעות בינה מלאכותית אחרי הכל, הנה דוגמה למימוש של המחלקה CheckCustomerHistory המנצלת עיבוד בינה מלאכותית באמצעות מודול ChatCompletion מספריית Raix שלי:

```
class CheckCustomerHistory
  include Raix::ChatCompletion

  attr_accessor :fraud_detection

  INSTRUCTION = <<~END
transaction customer's a checking with tasked assistant AI an are You
transaction current the Given indicators. fraud potential for history
any identify to data the analyze ,transactions past customer's the and
anomalies. or patterns suspicious

transaction ,transactions of frequency the as such factors Consider
customer's the from deviations any and ,locations geographical ,amounts
range the in float a as score probability a generate to behavior typical
fraud). of certainty absolute being 1 (with 1 to 0 of

areas or flags red any highlighting ,analysis your of results the Output
format: JSON following the in concern of

} <Float> probability: ,findings> your of <Summary description: {
END

  def self.check(fraud_detection)
    new(fraud_detection).call
  end

  def call
    chat_completion(json: true).tap do |result|
      fraud_detection.add_risk_factor(**result)
    end
    Result.ok(fraud_detection)
  rescue StandardError => e
    Result.err(FraudDetectionError.new(e))
  end

  private

  def initialize(fraud_detection)
    self.fraud_detection = fraud_detection
  end

  def transcript
```

```
43      tx_history = fraud_detection.transaction.user.tx_history
44      [
45        { system: INSTRUCTION },
46        { user: " history: Transaction#{tx_history.to_json}" },
47        { assistant: "transaction. current the provide Please OK." },
48        { user: " transaction: Current#{fraud_detection.transaction.to_json}" }
49      ]
50    end
51  end
```

בדוגמה זו, המחלקה CheckCustomerHistory מגדירה קבוע INSTRUCTION המספק הוראות ספציפיות למודל הבינה המלאכותית כיצד לנתח את היסטוריית העסקאות של הלקוח לזיהוי סממני הונאה באמצעות הנחיית מערכת

המתודה self.check היא מתודת מחלקה שמאתחלת מופע חדש של CheckCustomerHistory עם אובייקט detection_fraud וקוראת למתודת call כדי לבצע את ניתוח היסטוריית הלקוח.

בתוך מתודת call, היסטוריית העסקאות של הלקוח מאוחזרת ומעוצבת לתמליל שמועבר למודל הבינה המלאכותית. מודל הבינה המלאכותית מנתח את היסטוריית העסקאות בהתבסס על ההוראות שסופקו ומחזיר סיכום של ממצאיו.

הממצאים מתווספים לאובייקט detection_fraud, והאובייקט המעודכן detection_fraud מוחזר כ-Result מוצלח.

באמצעות שימוש במודול ChatCompletion, המחלקה CheckCustomerHistory יכולה לנצל את כוח הבינה המלאכותית כדי לנתח את היסטוריית העסקאות של הלקוח ולזהות סממני הונאה פוטנציאליים. זה מאפשר טכניקות מתוחכמות ומסתגלות יותר לזיהוי הונאות, כאשר מודל הבינה המלאכותית יכול ללמוד ולהסתגל לדפוסים וחריגות חדשים לאורך זמן.

ה-FraudDetectionWorker המעודכן והמחלקה CheckCustomerHistory מדגימים כיצד ניתן לשלב בצורה חלקה עובדי בינה מלאכותית, תוך שיפור תהליך זיהוי ההונאות באמצעות יכולות ניתוח וקבלת החלטות חכמות.

ניתוח רגשות לקוח

הנה עוד דוגמה דומה לאופן שבו ניתן ליישם עובד לניתוח רגשות לקוח. הפעם עם הרבה פחות הסברים, מכיוון שאתם כבר אמורים להבין את אופן הפעולה של סגנון תכנות זה:

```
class CustomerSentimentAnalysisWorker
  include Wisper::Publisher

  def call(feedback)
    Result.ok(feedback)
      .and_then(PreprocessFeedback.method(:preprocess))
      .map(PerformSentimentAnalysis.method(:analyze))
      .map(ExtractKeyPhrases.method(:extract))
      .map(IdentifyTrends.method(:identify))
      .map(GenerateInsights.method(:generate)).then do |result|

      case result
      in { err: SentimentAnalysisError => error }
        Honeybadger.notify(error.message, context: {feedback:})
      in { ok: SentimentAnalysisResult => result }
        broadcast(:sentiment_analysis_completed, result)
      end
    end
  end
end
```

בדוגמה זו, ה-CustomerSentimentAnalysisWorker כולל שלבים של עיבוד מקדים של המשוב (למשל, הסרת רעש, פירוק למילים), ביצוע ניתוח רגשות לקביעת הרגש הכללי (חיובי, שלילי או ניטרלי), חילוץ ביטויי מפתח ונושאים, זיהוי מגמות ודפוסים, ויצירת תובנות פעילות על בסיס הניתוח.

יישומים בתחום הבריאות

בתחום הבריאות, עובדי בינה מלאכותית יכולים לסייע לאנשי מקצוע רפואיים וחוקרים במגוון משימות, המובילות לשיפור בתוצאות הטיפול במטופלים והאצת תגליות רפואיות. להלן מספר דוגמאות:

קליטת מטופלים

עובדי בינה מלאכותית יכולים לייעל את תהליך קליטת המטופלים על ידי אוטומציה של משימות שונות ומתן סיוע חכם.

תיאום תורים: עובדי בינה מלאכותית יכולים לטפל בתיאום תורים תוך הבנת העדפות המטופל, זמינותו, ודחיפות צרכיו הרפואיים. הם יכולים לתקשר עם מטופלים באמצעות ממשקי שיחה, להדריך אותם בתהליך קביעת התור ולמצוא את המועדים המתאימים ביותר בהתבסס על דרישות המטופל וזמינות ספק שירותי הבריאות.

איסוף היסטוריה רפואית: במהלך קליטת המטופל, עובדי בינה מלאכותית יכולים לסייע באיסוף ותיעוד ההיסטוריה הרפואית של המטופל. הם יכולים לנהל דיאלוג אינטראקטיבי עם מטופלים, לשאול שאלות רלוונטיות על מצבים רפואיים קודמים, תרופות, אלרגיות והיסטוריה משפחתית. עובדי הבינה המלאכותית יכולים להשתמש בטכניקות עיבוד שפה טבעית לפירוש ומבנוי המידע שנאסף, תוך הבטחת תיעוד מדויק ברשומה הרפואית הממוחשבת של המטופל.

הערכת תסמינים וסיווג: עובדי בינה מלאכותית יכולים לבצע הערכות תסמינים ראשוניות על ידי תשאול המטופלים לגבי התסמינים הנוכחיים, משכם, חומרתם וגורמים נלווים. באמצעות שימוש במאגרי ידע רפואיים ומודלים של למידת מכונה, עובדים אלה יכולים לנתח את המידע שסופק ולייצר אבחנות מבדלות ראשוניות או להמליץ על צעדים מתאימים הבאים, כגון קביעת פגישת ייעוץ עם ספק שירותי בריאות או הצעת אמצעי טיפול עצמי.

אימות ביטוח: עובדי בינה מלאכותית יכולים לסייע באימות ביטוח במהלך קליטת המטופל. הם יכולים לאסוף פרטי ביטוח מהמטופל, לתקשר עם ספקי הביטוח באמצעות ממשקי API או שירותי אינטרנט, ולאמת זכאות וכיסוי. אוטומציה זו מסייעת לייעל את תהליך אימות הביטוח, מפחיתה עומס מנהלי ומבטיחה קליטת מידע מדויק.

הדרכת מטופלים והוראות: עובדי בינה מלאכותית יכולים לספק למטופלים חומרי הדרכה והוראות רלוונטיים בהתאם למצבם הרפואי הספציפי או להליכים העתידיים. הם יכולים להעביר תוכן מותאם אישית, לענות על שאלות נפוצות, ולהציע הדרכה לגבי הכנות לפני פגישה, הוראות לנטילת תרופות, או טיפול לאחר הליך רפואי. זה עוזר לשמור על מטופלים מיודעים ומעורבים לאורך כל מסע הטיפול הרפואי שלהם.

באמצעות שילוב עובדי בינה מלאכותית בתהליך קבלת המטופלים, ארגוני בריאות יכולים לשפר את היעילות, להפחית זמני המתנה ולשפר את חווית המטופל הכוללת. עובדים אלה יכולים לטפל במשימות שגרתיות, לאסוף מידע מדויק ולספק סיוע מותאם אישית, מה שמאפשר לאנשי המקצוע בתחום הבריאות להתמקד במתן טיפול איכותי למטופלים.

הערכת סיכונים למטופל

עובדי בינה מלאכותית יכולים למלא תפקיד מכריע בהערכת סיכוני מטופלים על ידי ניתוח מקורות מידע שונים ויישום טכניקות אנליטיות מתקדמות.

שילוב נתונים: עובדי בינה מלאכותית יכולים לאסוף ולהבין נתוני מטופלים ממקורות מרובים, כגון רשומות רפואיות ממוחשבות, דימות רפואי, תוצאות מעבדה, מכשירים לבישים וגורמים חברתיים המשפיעים על הבריאות. על ידי איחוד מידע זה לפרופיל מטופל מקיף, עובדי הבינה המלאכותית יכולים לספק תמונה הוליסטית של מצב הבריאות וגורמי הסיכון של המטופל.

שכבות סיכון: עובדי בינה מלאכותית יכולים להשתמש במודלים חיזויים כדי לשייך מטופלים לקטגוריות סיכון שונות על בסיס המאפיינים האישיים ונתוני הבריאות שלהם. שכבות סיכון אלה מאפשרות לספקי שירותי בריאות לתעדף מטופלים הזקוקים לתשומת לב או התערבות מיידית יותר. לדוגמה, מטופלים שזוהו כבעלי סיכון גבוה למצב מסוים יכולים להיות מסומנים למעקב צמוד יותר, אמצעי מניעה או התערבות מוקדמת.

פרופילי סיכון מותאמים אישית: עובדי בינה מלאכותית יכולים ליצור פרופילי סיכון מותאמים אישית לכל מטופל, תוך הדגשת הגורמים הספציפיים התורמים לציוני הסיכון שלהם. פרופילים אלה יכולים לכלול תובנות לגבי אורח החיים של המטופל, נטיות גנטיות, גורמים סביבתיים וגורמים חברתיים המשפיעים על הבריאות. על ידי מתן פירוט מפורט של גורמי סיכון, עובדי בינה מלאכותית יכולים לסייע לספקי שירותי בריאות להתאים אסטרטגיות מניעה ותוכניות טיפול לצרכים האישיים של המטופל.

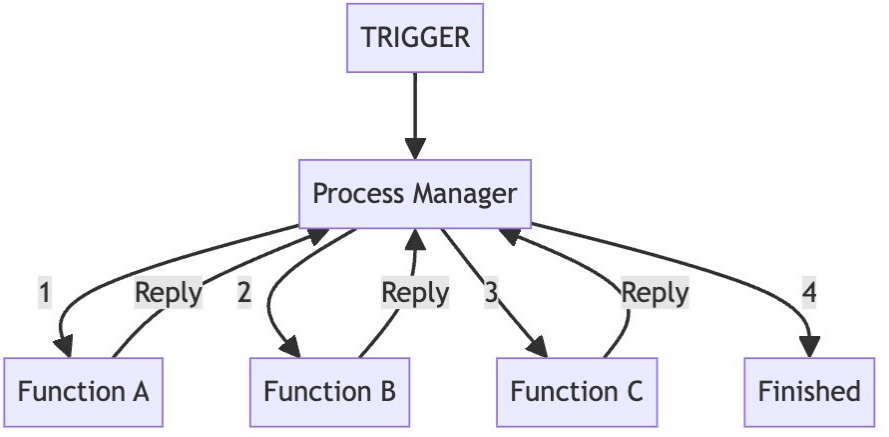
ניטור סיכונים מתמשך: עובדי בינה מלאכותית יכולים לנטר ברציפות נתוני מטופלים ולעדכן הערכות סיכון בזמן אמת. כאשר מידע חדש זמין, כגון שינויים בסימנים חיוניים, תוצאות מעבדה או היענות לתרופות, עובדי הבינה המלאכותית יכולים לחשב מחדש ציוני סיכון ולהתריע לספקי שירותי בריאות על שינויים משמעותיים. ניטור פרואקטיבי זה מאפשר התערבויות במועד והתאמות לתוכניות הטיפול במטופל.

תמיכה בקבלת החלטות קליניות: עובדי בינה מלאכותית יכולים לשלב תוצאות הערכת סיכונים במערכות תמיכה בקבלת החלטות קליניות, תוך מתן המלצות והתראות מבוססות ראיות לספקי שירותי בריאות. לדוגמה, אם ציון הסיכון של מטופל למצב מסוים עובר סף מסוים, עובד הבינה המלאכותית יכול להנחות את ספק שירותי הבריאות לשקול בדיקות

אבחון ספציפיות, אמצעי מניעה או אפשרויות טיפול המבוססות על הנחיות קליניות ושיטות עבודה מיטביות.

עובדים אלה יכולים לעבד כמויות עצומות של נתוני מטופלים, ליישם ניתוחים מתוחכמים ולייצר תובנות פעילות לתמיכה בקבלת החלטות קליניות. זה מוביל בסופו של דבר לשיפור בתוצאות המטופלים, הפחתת עלויות הבריאות ושיפור ניהול בריאות האוכלוסייה.

תוכנת עובד AI כמנהל תהליכים



בהקשר של יישומים מבוססי AI, ניתן לתכנן תוכנת עובד כך שתתפקד כמנהל תהליכים , כפי שמתואר בספר "תבניות אינטגרציה ארגונית" מאת Hohpe Gregor . מנהל תהליכים הוא רכיב מרכזי השומר על מצב התהליך וקובע את שלבי העיבוד הבאים בהתבסס על תוצאות ביניים.

כאשר תוכנת עובד AI פועלת כמנהל תהליכים, היא מקבלת הודעה נכנסת המאתחלת את התהליך, הידועה כהודעת הפעלה . תוכנת העובד AI שומרת אז על מצב ביצוע התהליך (כתמליל שיחה) ומטפלת בהודעה באמצעות סדרה של שלבי עיבוד המיושמים כפונקציות כלי, אשר יכולות להיות רציפות או מקבילות, ונקראות לפי שיקול דעתה.

אם אתה משתמש בסוג של מודל AI כמו GPT-4 שיודע לבצע פונקציות במקביל, אז תוכנת העובד שלך יכולה לבצע מספר שלבים בו-זמנית. אני מודה שלא ניסיתי לעשות זאת בעצמי ותחושת הבטן שלי אומרת שהתוצאות עשויות להשתנות.

לאחר כל שלב עיבוד בודד, השליטה חוזרת לתוכנת העובד, AI, מה שמאפשר לה לקבוע את שלב(י) העיבוד הבא(ים) בהתבסס על המצב הנוכחי והתוצאות שהתקבלו.

אחסן את הודעות ההפעלה שלך

מניסיוני, חכם ליישם את הודעת ההפעלה שלך כאובייקט מגובה מסד נתונים . כך כל מופע של תהליך מזוהה על ידי מפתח ראשי ייחודי ומספק לך מקום לאחסון המצב הקשור לביצוע, כולל תמליל השיחה של ה-AI.

לדוגמה, הנה גרסה מפושטת של מחלקת המודל AccountChange של Olympia, המייצגת בקשה לביצוע שינוי בחשבון משתמש.

```
1  \nformation Schema == #
2  #
3  account_changes name: Table #
4  #
5  key primary ,null not          :uuid       id  #
6  :string description  #
7  null not         :string      state  #
8  :jsonb  transcript  #
9  null not       :datetime  created_at  #
10 null not       :datetime  updated_at  #
11 null not          :uuid  account_id  #
12 #
13 \ndexes #
14 #
15 (account_id)  index_account_changes_on_account_id  #
16 #
17 Keys Foreign #
18 #
19 accounts.id) => (account_id  fk_rails_... #
20 #
21 class AccountChange < ApplicationRecord
```

```
    belongs_to :account

    validates :description, presence: true

    after_commit -> {
      broadcast(:account_change_requested, self)
    }, on: :create

    state_machine initial: :requested do
      event :completed do
        transition all => :complete
      end
      event :failed do
        transition all => :requires_human_review
      end
    end
  end
```

מחלקת AccountChange משמשת כהודעת טריגר המפעילה תהליך לטיפול בבקשת שינוי החשבון. שימו לב כיצד היא משודרת למערכת הפרסום/הרשמה מבוססת Wisper של Olympia לאחר שפעולת היצירה מסיימת להתבצע.

שמירת הודעת הטריגר במסד הנתונים בצורה זו מספקת רישום קבוע של כל בקשת שינוי חשבון. לכל מופע של מחלקת AccountChange מוקצה מפתח ראשי ייחודי, המאפשר זיהוי ומעקב קל אחר בקשות בודדות. הדבר שימושי במיוחד למטרות תיעוד ביקורת, מכיוון שהוא מאפשר למערכת לשמור על רישום היסטורי של כל שינויי החשבון, כולל מתי הם התבקשו, אילו שינויים התבקשו, והמצב הנוכחי של כל בקשה.

בדוגמה הנתונה, מחלקת AccountChange כוללת שדות כמו description ללכידת פרטי השינוי המבוקש, state לייצוג המצב הנוכחי של הבקשה (למשל, התבקש, הושלם, דורש_בדיקת_אדם), ו-transcript לשמירת תמליל השיחה של הבינה המלאכותית הקשור לבקשה. שדה ה-description הוא הפקודה בפועל המשמשת להתחלת השלמת הצ'אט הראשונה עם הבינה המלאכותית. שמירת נתונים אלה מספקת הקשר חשוב ומאפשרת מעקב וניתוח טובים יותר של תהליך שינוי החשבון.

שמירת הודעות טריגר במסד הנתונים מאפשרת טיפול חזק בשגיאות והתאוששות. אם מתרחשת שגיאה במהלך עיבוד בקשת שינוי חשבון, המערכת מסמנת את הבקשה ככשלה

ומעבירה אותה למצב הדורש התערבות אנושית. זה מבטיח שאף בקשה לא תאבד או תישכח, וכל בעיה יכולה להיות מטופלת ונפתרת כראוי.

עובד הבינה המלאכותית, כמנהל תהליכים, מספק נקודת שליטה מרכזית ומאפשר יכולות חזקות של דיווח ותיקון באגים בתהליך. עם זאת, חשוב לציין שהשימוש בעובד בינה מלאכותית כמנהל תהליכים עבור כל תרחיש זרימת עבודה באפליקציה שלך עלול להיות מוגזם.

שילוב עובדי בינה מלאכותית בארכיטקטורת האפליקציה שלך

בעת שילוב עובדי בינה מלאכותית בארכיטקטורת האפליקציה שלך, יש להתייחס למספר שיקולים טכניים כדי להבטיח אינטגרציה חלקה ותקשורת יעילה בין עובדי הבינה המלאכותית לבין רכיבי האפליקציה האחרים. חלק זה דן בהיבטים מרכזיים של תכנון ממשקים אלה, טיפול בזרימת נתונים, וניהול מחזור החיים של עובדי בינה מלאכותית.

תכנון ממשקים ופרוטוקולי תקשורת ברורים

כדי לאפשר אינטגרציה חלקה בין עובדי בינה מלאכותית לרכיבי אפליקציה אחרים, חיוני להגדיר ממשקים ופרוטוקולי תקשורת ברורים. שקלו את הגישות הבאות:

אינטגרציה מבוססת API: חשיפת הפונקציונליות של עובדי AI באמצעות ממשקי API מוגדרים היטב, כגון נקודות קצה RESTful או סכמות GraphQL. זה מאפשר לרכיבים אחרים לתקשר עם עובדי ה-AI באמצעות בקשות ותגובות HTTP סטנדרטיות. אינטגרציה מבוססת API מספקת חוזה ברור בין עובדי ה-AI לבין הרכיבים הצורכים, מה שמקל על הפיתוח, הבדיקה והתחזוקה של נקודות האינטגרציה.

תקשורת מבוססת הודעות: יישום דפוסי תקשורת מבוססי הודעות, כגון תורי הודעות או מערכות פרסום-הרשמה, כדי לאפשר אינטראקציה אסינכרונית בין עובדי AI לרכיבים אחרים.

גישה זו מנתקת את עובדי ה-AI משאר האפליקציה, ומאפשרת יכולת הרחבה טובה יותר, עמידות בפני תקלות וצימוד רופף. תקשורת מבוססת הודעות שימושית במיוחד כאשר העיבוד שמבוצע על ידי עובדי AI צורך זמן רב או משאבים רבים, מכיוון שהיא מאפשרת לחלקים אחרים של האפליקציה להמשיך לפעול מבלי להמתין להשלמת המשימות של עובדי ה-AI.

ארכיטקטורה מונחית אירועים: תכנון המערכת סביב אירועים וטריגרים המפעילים עובדי AI כאשר מתקיימים תנאים ספציפיים. עובדי AI יכולים להירשם לאירועים רלוונטיים ולהגיב בהתאם, מבצעים את המשימות המיועדות להם כאשר האירועים מתרחשים. ארכיטקטורה מונחית אירועים מאפשרת עיבוד בזמן אמת ומאפשרת הפעלת עובדי AI לפי דרישה, מה שמפחית צריכת משאבים מיותרת. גישה זו מתאימה היטב לתרחישים בהם עובדי AI צריכים להגיב לפעולות ספציפיות או שינויים במצב האפליקציה.

טיפול בזרימת נתונים וסנכרון

בעת שילוב עובדי AI באפליקציה שלך, חשוב להבטיח זרימת נתונים חלקה ולשמור על עקביות הנתונים בין עובדי ה-AI לרכיבים אחרים. שקול את ההיבטים הבאים:

הכנת נתונים: לפני הזנת נתונים לעובדי ,AI ייתכן שתצטרך לבצע משימות הכנת נתונים שונות, כגון ניקוי, עיצוב ו/או המרת נתוני הקלט. אתה לא רק רוצה לוודא שעובדי ה-AI יכולים לעבד ביעילות, אלא גם לוודא שאינך מבזבז טוקנים בהקדשת תשומת לב למידע שהעובד עשוי לראות כחסר תועלת במקרה הטוב, או מסיח במקרה הגרוע. הכנת נתונים עשויה לכלול משימות כמו הסרת רעש, טיפול בערכים חסרים, או המרת סוגי נתונים.

התמדת נתונים: כיצד תאחסן ותשמר את הנתונים הזורמים אל ומחוץ לעובדי AI? שקול גורמים כמו נפח הנתונים, דפוסי שאילתות, ויכולת הרחבה. האם אתה צריך לשמור את תמליל ה-AI כשיקוף של "תהליך החשיבה" שלו למטרות ביקורת או ניפוי באגים, או שמספיק לשמור רשומה של התוצאות בלבד?

אחזור נתונים: השגת הנתונים הנדרשים לעובדים עשויה לכלול שאילתות למסדי נתונים, קריאה מקבצים או גישה ל-API חיצוניים. חשוב להתחשב בזמני התגובה ובאופן שבו עובדי הבינה המלאכותית יקבלו גישה לנתונים העדכניים ביותר. האם הם זקוקים לגישה מלאה למסד הנתונים שלך או שעליך להגדיר את היקף הגישה שלהם באופן מצומצם בהתאם למה

שהם עושים? ומה לגבי הרחבה? שקול מנגנוני מטמון לשיפור הביצועים והפחתת העומס על מקורות הנתונים.

סנכרון נתונים: כאשר רכיבים מרובים, כולל עובדי בינה מלאכותית, ניגשים ומשנים נתונים משותפים, חשוב ליישם מנגנוני סנכרון מתאימים כדי לשמור על עקביות הנתונים. אסטרטגיות נעילת מסד נתונים, כגון נעילה אופטימית או פסימית, עשויות לעזור למנוע התנגשויות ולהבטיח את שלמות הנתונים. יש ליישם טכניקות ניהול טרנזקציות לקיבוץ פעולות נתונים קשורות ולשמירה על תכונות אטומיות, עקביות, בידוד, ועמידות (ACID)

טיפול בשגיאות והתאוששות: יש ליישם מנגנוני טיפול בשגיאות והתאוששות חזקים כדי להתמודד עם בעיות הקשורות לנתונים שעלולות להתעורר במהלך תהליך זרימת הנתונים. יש לטפל בחריגים בצורה מסודרת ולספק הודעות שגיאה משמעותיות שיסייעו בניפוי באגים. יש ליישם מנגנוני ניסיון חוזר ואסטרטגיות גיבוי כדי להתמודד עם כשלים זמניים או הפרעות רשת. יש להגדיר נהלים ברורים לשחזור והחזרת נתונים במקרה של שחיתות או אובדן נתונים.

באמצעות תכנון ויישום קפדני של מנגנוני זרימת נתונים וסנכרון, תוכל להבטיח שלעובדי הבינה המלאכותית שלך תהיה גישה לנתונים מדויקים, עקביים ומעודכנים. זה מאפשר להם לבצע את משימותיהם ביעילות ולהפיק תוצאות אמינות.

ניהול מחזור החיים של עובדי בינה מלאכותית

פתח תהליך סטנדרטי לאתחול והגדרת עובדי בינה מלאכותית. אני מעדיף מסגרות עבודה המתקננות את האופן שבו מגדירים הגדרות כגון שמות מודלים, הנחיות מערכת והגדרות פונקציות. ודא שתהליך האתחול הוא אוטומטי וניתן לשחזור כדי להקל על הפריסה וההרחבה.

יישם מנגנוני ניטור ותיעוד מקיפים למעקב אחר תקינות וביצועי עובדי הבינה המלאכותית. אסוף מדדים כגון ניצול משאבים, זמן עיבוד, שיעורי שגיאה ותפוקה. השתמש במערכות תיעוד מרכזיות כמו ELK stack (Elasticsearch, Logstash, Kibana) כדי לאגד ולנתח יומנים ממספר עובדי בינה מלאכותית.

בנה סבילות לתקלות וחוסן לתוך ארכיטקטורת עובד ה-AI. יישם מנגנוני טיפול בשגיאות והתאוששות כדי להתמודד באופן חלק עם כשלים או חריגות. מודלים שפתיים גדולים הם

עדיין טכנולוגיה חדשנית; ספקים נוטים ליפול לעתים קרובות בזמנים בלתי צפויים. השתמש במנגנוני ניסיון חוזר ומפסקי זרם כדי למנוע כשלים מצטברים.

יכולת הרכבה ותזמור של עובדי AI

אחד היתרונות המרכזיים של ארכיטקטורת עובד ה-AI היא יכולת ההרכבה שלה, המאפשרת לך לשלב ולתזמר מספר עובדי AI כדי לפתור בעיות מורכבות. על ידי פירוק משימה גדולה יותר למשימות משנה קטנות יותר וניתנות לניהול, כל אחת מטופלת על ידי עובד AI מתמחה, אתה יכול ליצור מערכות חזקות וגמישות. בחלק זה, נחקור גישות שונות להרכבה ותזמור של "ריבוי" עובדי AI.

שרשור עובדי AI לתהליכי עבודה מרובי שלבים

במקרים רבים, ניתן לפרק משימה מורכבת לסדרה של צעדים רציפים, כאשר הפלט של עובד AI אחד הופך לקלט עבור הבא. שרשור זה של עובדי AI יוצר תהליך עבודה או צינור מרובה שלבים. כל עובד AI בשרשרת מתמקד במשימת משנה ספציפית, והפלט הסופי הוא תוצאה של המאמצים המשולבים של כל העובדים.

הבה נבחן דוגמה בהקשר של יישום Rails on Ruby לעיבוד תוכן שנוצר על ידי משתמשים. תהליך העבודה כולל את השלבים הבאים, שככל הנראה כל אחד מהם פשוט מדי מכדי שיהיה שווה לפרק אותו בצורה כזו במקרי שימוש אמיתיים, אבל הם הופכים את הדוגמה לקלה יותר להבנה:

1. **ניקוי טקסט:** עובד AI האחראי על הסרת תגיות HTML, המרת טקסט לאותיות קטנות, וטיפול בנורמליזציה של יוניקוד.
2. **זיהוי שפה:** עובד AI המזהה את השפה של הטקסט המנוקה.
3. **ניתוח רגשות:** עובד AI הקובע את הרגש (חיובי, שלילי או ניטרלי) של הטקסט בהתבסס על השפה שזוהתה.
4. **קטגוריזציה של תוכן:** עובד AI המסווג את הטקסט לקטגוריות מוגדרות מראש באמצעות טכניקות עיבוד שפה טבעית.

הנה דוגמה מאוד מפושטת כיצד ניתן לשרשר את עובדי ה-AI הללו יחד באמצעות Ruby:

```ruby
class ContentProcessor
  def initialize(text)
    @text = text
  end

  def process
    cleaned_text = TextCleanupWorker.new(@text).call
    language = LanguageDetectionWorker.new(cleaned_text).call
    sentiment = SentimentAnalysisWorker.new(cleaned_text, language).call
    category = CategorizationWorker.new(cleaned_text, language).call

    { cleaned_text:, language:, sentiment:, category: }
  end
end
```

בדוגמה זו, המחלקה ContentProcessor מאותחלת עם הטקסט הגולמי ומשרשרת את עובדי הבינה המלאכותית יחד בשיטת process. כל עובד בינה מלאכותית מבצע את המשימה הספציפית שלו ומעביר את התוצאה לעובד הבא בשרשרת. הפלט הסופי הוא מבנה נתונים מסוג hash המכיל את הטקסט המנוקה, השפה שזוהתה, הרגש והקטגוריה של התוכן.

עיבוד מקבילי עבור עובדי בינה מלאכותית עצמאיים

בדוגמה הקודמת, עובדי הבינה המלאכותית משורשרים ברצף, כאשר כל עובד מעבד את הטקסט ומעביר את התוצאה לעובד הבא. עם זאת, אם יש לך מספר עובדי בינה מלאכותית שיכולים לפעול באופן עצמאי על אותו קלט, תוכל למטב את זרימת העבודה על ידי עיבודם במקביל.

בתרחיש הנתון, לאחר שניקוי הטקסט מתבצע על ידי TextCleanupWorker, העובדים LanguageDetectionWorker, SentimentAnalysisWorker, ו-CategorizationWorker יכולים כולם לעבד את הטקסט המנוקה באופן עצמאי. על ידי הרצת עובדים אלה במקביל, תוכל להפחית את זמן העיבוד הכולל ולשפר את היעילות של זרימת העבודה שלך.

כדי להשיג עיבוד מקבילי ב-Ruby, תוכל לנצל טכניקות מקביליות כמו תהליכונים או תכנות אסינכרוני. הנה דוגמה כיצד תוכל לשנות את המחלקה ContentProcessor כדי לעבד את שלושת העובדים האחרונים במקביל באמצעות תהליכונים:

```ruby
require 'concurrent'

class ContentProcessor
  def initialize(text)
    @text = text
  end

  def process
    cleaned_text = TextCleanupWorker.new(@text).call

    language_future = Concurrent::Future.execute do
      LanguageDetectionWorker.new(cleaned_text).call
    end

    sentiment_future = Concurrent::Future.execute do
      SentimentAnalysisWorker.new(cleaned_text).call
    end

    category_future = Concurrent::Future.execute do
      CategorizationWorker.new(cleaned_text).call
    end

    language = language_future.value
    sentiment = sentiment_future.value
    category = category_future.value

    { cleaned_text:, language:, sentiment:, category: }
  end
end
```

בגרסה המשופרת הזו, אנחנו משתמשים בספריית concurrent-ruby כדי ליצור אובייקטי Concurrent::Future עבור כל אחד מעובדי הבינה המלאכותית העצמאיים. Future מייצג חישוב שיתבצע באופן אסינכרוני בתהליכון נפרד.

לאחר שלב ניקוי הטקסט, אנו יוצרים שלושה אובייקטי Future: future_language, future_sentiment, ו-future_category. כל Future מפעיל את עובד הבינה המלאכותית המתאים לו (LanguageDetectionWorker, SentimentAnalysisWorker, ו-CategorizationWorker) בתהליכון נפרד, תוך העברת text_cleaned כקלט.

על ידי קריאה למתודה value על כל Future, אנחנו ממתינים להשלמת החישוב ומקבלים את

התוצאה. המתודה value חוסמת עד שהתוצאה זמינה, ומבטיחה שכל העובדים המקבילים סיימו את העיבוד לפני שממשיכים.

לבסוף, אנו בונים את מילון הפלט עם הטקסט המנוקה והתוצאות מהעובדים המקבילים, בדיוק כמו בדוגמה המקורית.

על ידי עיבוד עובדי הבינה המלאכותית העצמאיים במקביל, ניתן להפחית את זמן העיבוד הכולל בהשוואה להרצתם ברצף. אופטימיזציה זו מועילה במיוחד כאשר מתמודדים עם משימות צורכות זמן או בעיבוד כמויות גדולות של נתונים.

עם זאת, חשוב לציין שהשיפור בביצועים בפועל תלוי בגורמים שונים, כמו מורכבות כל עובד, משאבי המערכת הזמינים, והתקורה של ניהול תהליכונים. תמיד מומלץ לבצע בדיקות ביצועים ופרופיילינג לקוד שלך כדי לקבוע את רמת המקביליות האופטימלית למקרה השימוש הספציפי שלך.

בנוסף, בעת יישום עיבוד מקבילי, יש להיות ערים למשאבים משותפים או תלויות בין העובדים. יש לוודא שהעובדים יכולים לפעול באופן עצמאי ללא התנגשויות או מצבי מרוץ. אם קיימות תלויות או משאבים משותפים, ייתכן שתצטרך ליישם מנגנוני סנכרון מתאימים כדי לשמור על שלמות הנתונים ולמנוע בעיות כמו קיפאון או תוצאות לא עקביות.

מנעול הפרשן הגלובלי של Ruby ועיבוד אסינכרוני

חשוב להבין את ההשלכות של מנעול הפרשן הגלובלי (GIL) של Ruby בעת שקילת עיבוד אסינכרוני מבוסס תהליכונים ב-Ruby.

ה-GIL הוא מנגנון בפרשן של Ruby המבטיח שרק תהליכון אחד יכול להריץ קוד Ruby בכל רגע נתון, גם במעבדים מרובי ליבות. המשמעות היא שבעוד שניתן ליצור ולנהל מספר תהליכונים בתוך תהליך Ruby, רק תהליכון אחד יכול להריץ קוד Ruby באופן פעיל בכל רגע נתון.

ה-GIL תוכנן כדי לפשט את היישום של מפרש Ruby ולספק בטיחות תהליכונים עבור מבני הנתונים הפנימיים של Ruby. עם זאת, הוא גם מגביל את הפוטנציאל להרצה מקבילית

אמיתית של קוד Ruby.

כאשר אתה משתמש בתהליכונים ב-Ruby, כמו עם ספריית concurrent-ruby או מחלקת ה-Thread המובנית, התהליכונים כפופים למגבלות ה-GIL. ה-GIL מאפשר לכל תהליכון להריץ קוד Ruby למשך פרק זמן קצר לפני שהוא עובר לתהליכון אחר, מה שיוצר אשליה של הרצה מקבילית.

עם זאת, בגלל ה-GIL, ההרצה בפועל של קוד Ruby נשארת רציפה. בזמן שתהליכון אחד מריץ קוד Ruby, תהליכונים אחרים למעשה מושהים, ממתינים לתורם לקבל את ה-GIL ולהתבצע.

משמעות הדבר היא שעיבוד אסינכרוני מבוסס תהליכונים ב-Ruby יעיל ביותר עבור משימות מוגבלות קלט/פלט, כמו המתנה לתגובות API חיצוניות (כגון מודלי שפה גדולים המאוחסנים על ידי צד שלישי) או ביצוע פעולות קלט/פלט של קבצים. כאשר תהליכון נתקל בפעולת קלט/פלט, הוא יכול לשחרר את ה-GIL, מה שמאפשר לתהליכונים אחרים לפעול בזמן ההמתנה להשלמת הקלט/פלט.

מצד שני, עבור משימות מוגבלות מעבד, כמו חישובים אינטנסיביים או עיבוד ממושך של עובדי בינה מלאכותית, ה-GIL עלול להגביל את פוטנציאל שיפור הביצועים של מקבילות מבוססת תהליכונים. מכיוון שרק תהליכון אחד יכול להריץ קוד Ruby בכל פעם, זמן ההרצה הכולל עשוי שלא להיות מופחת משמעותית בהשוואה לעיבוד רציף.

כדי להשיג הרצה מקבילית אמיתית עבור משימות מוגבלות מעבד ב-Ruby, ייתכן שתצטרך לחקור גישות חלופיות, כגון:

- שימוש במקבילות מבוססת תהליכים עם מספר תהליכי Ruby, כל אחד רץ על ליבת מעבד נפרדת.
- ניצול ספריות חיצוניות או מסגרות עבודה המספקות הרחבות native או ממשקים לשפות ללא GIL, כמו C או Rust.,
- שימוש במסגרות מחשוב מבוזר או תורי הודעות כדי לחלק משימות בין מכונות או תהליכים מרובים.

חשוב לשקול את אופי המשימות שלך ואת המגבלות שמטיל ה-GIL בעת תכנון ויישום של עיבוד אסינכרוני ב-Ruby. בעוד שעיבוד אסינכרוני מבוסס תהליכונים יכול לספק יתרונות

עבור משימות מוגבלות קלט/פלט, הוא עשוי שלא להציע שיפורי ביצועים משמעותיים עבור משימות מוגבלות מעבד בגלל מגבלות ה-GIL.

טכניקות אנסמבל לשיפור דיוק

טכניקות אנסמבל כוללות שילוב של פלטים ממספר עובדי בינה מלאכותית כדי לשפר את הדיוק הכולל או את החוסן של המערכת. במקום להסתמך על עובד בינה מלאכותית יחיד, טכניקות אנסמבל מנצלות את האינטליגנציה הקולקטיבית של מספר עובדים כדי לקבל החלטות מושכלות יותר.

אנסמבלים חשובים במיוחד כאשר חלקים שונים בתהליך העבודה שלך עובדים טוב יותר עם מודלים שונים של בינה מלאכותית, דבר שנפוץ יותר ממה שאתה עשוי לחשוב. מודלים חזקים כמו GPT-4 יקרים מאוד בהשוואה לאפשרויות קוד פתוח פחות מתקדמות, וכנראה אינם נחוצים עבור כל שלב בתהליך העבודה של האפליקציה שלך.

טכניקת אנסמבל נפוצה היא הצבעת רוב , שבה מספר עובדי בינה מלאכותית מעבדים באופן עצמאי את אותו קלט, והפלט הסופי נקבע על פי הסכמת הרוב. גישה זו יכולה לעזור בהפחתת ההשפעה של שגיאות עובדים בודדים ולשפר את האמינות הכוללת של המערכת.

הבה נתבונן בדוגמה שבה יש לנו שלושה עובדי בינה מלאכותית לניתוח רגשות , כל אחד משתמש במודל שונה או מקבל הקשרים שונים. אנחנו יכולים לשלב את הפלטים שלהם באמצעות הצבעת רוב כדי לקבוע את תחזית הרגש הסופית.

```ruby
class SentimentAnalysisEnsemble
  def initialize(text)
    @text = text
  end

  def analyze
    predictions = [
      SentimentAnalysisWorker1.new(@text).analyze,
      SentimentAnalysisWorker2.new(@text).analyze,
      SentimentAnalysisWorker3.new(@text).analyze
    ]

    predictions
      .group_by { |sentiment| sentiment }
      .max_by { |_, votes| votes.size }
      .first

  end
end
```

בדוגמה זו, המחלקה SentimentAnalysisEnsemble. מאותחלת עם הטקסט ומפעילה שלושה עובדי בינה מלאכותית שונים לניתוח רגשות. המתודה analyze אוספת את התחזיות מכל עובד וקובעת את הרגש הרוב באמצעות המתודות by_group ו-by_max. הפלט הסופי הוא הרגש שמקבל את מירב הקולות מאנסמבל העובדים

אנסמבלים הם בבירור מקרה שבו כדאי להשקיע זמן בניסויים עם מקביליות.

בחירה והפעלה דינמית של עובדי בינה מלאכותית

במקרים מסוימים, אם לא ברובם, העובד הספציפי של הבינה המלאכותית שיופעל עשוי להיות תלוי בתנאי זמן ריצה או בקלט המשתמש. בחירה והפעלה דינמית של עובדי בינה מלאכותית מאפשרות גמישות והסתגלות במערכת.

ייתכן שתמצאו את עצמכם מתפתים לנסות להכניס פונקציונליות רבה לעובד בינה מלאכותית יחיד, תוך מתן פונקציות רבות ופרומפט מסובך שמסביר כיצד להשתמש בהן. התנגדו לפיתוי, האמינו לי. אחת הסיבות שהגישה שאנו דנים בה בפרק זה נקראת "ריבוי עובדים" היא להזכיר לנו שרצוי שיהיו הרבה עובדים מתמחים, כשכל אחד מבצע את תפקידו הקטן לשירות המטרה הגדולה יותר.

לדוגמה, שקלו אפליקציית צ'אטבוט שבה עובדי בינה מלאכותית שונים אחראים לטיפול בסוגים שונים של שאילתות משתמש. בהתבסס על קלט המשתמש, האפליקציה בוחרת באופן דינמי את עובד הבינה המלאכותית המתאים לעיבוד השאילתה.

```
class ChatbotController < ApplicationController
  def process_query
    query = params[:query]
    query_type = QueryClassifierWorker.new(query).classify

    case query_type
    when 'greeting'
      response = GreetingWorker.new(query).generate_response
    when 'product_inquiry'
      response = ProductInquiryWorker.new(query).generate_response
    when 'order_status'
      response = OrderStatusWorker.new(query).generate_response
    else
      response = DefaultResponseWorker.new(query).generate_response
    end

    render json: { response: response }
  end
end
```

בדוגמה זו, ה-ChatbotController מקבל שאילתת משתמש דרך פעולת ה-query_process. תחילה הוא משתמש ב-QueryClassifierWorker כדי לקבוע את סוג השאילתה. בהתבסס על סוג השאילתה המסווגת, הבקר בוחר באופן דינמי את עובד הבינה המלאכותית המתאים ליצירת התגובה. בחירה דינמית זו מאפשרת לצ'אטבוט לטפל בסוגים שונים של שאילתות ולנתב אותן לעובדי הבינה המלאכותית הרלוונטיים.

מכיוון שהעבודה של ה-QueryClassifierWorker היא יחסית פשוטה ואינה דורשת הרבה הקשר או הגדרות פונקציות, סביר להניח שתוכל ליישם אותה באמצעות מודל LLM קטן ומהיר במיוחד כמו mistralai/mixtral-8x7b-instruct:nitro. יש לו יכולות שמתקרבות לרמת GPT-4 במשימות רבות, ובזמן כתיבת שורות אלה, Groq יכולה לשרת אותו בתפוקה מדהימה של 444 טוקנים לשנייה.

שילוב עיבוד שפה טבעית מסורתי עם מודלי שפה גדולים

בעוד שמודלי שפה גדולים (LLMs) חוללו מהפכה בתחום עיבוד השפה הטבעית (NLP), ומציעים גמישות וביצועים חסרי תקדים במגוון רחב של משימות, הם לא תמיד הפתרון היעיל ביותר או המשתלם ביותר מבחינה כלכלית לכל בעיה. במקרים רבים, שילוב טכניקות עיבוד שפה טבעית מסורתיות עם מודלי שפה גדולים יכול להוביל לגישות יותר ממוקדות, מותאמות וחסכוניות לפתרון אתגרי עיבוד שפה טבעית ספציפיים.

חשבו על מודלי שפה גדולים כסכין שוויצרי של עיבוד שפה טבעית - חזקים וורסטיליים להפליא, אך לא בהכרח הכלי הטוב ביותר לכל משימה. לעתים, כלי ייעודי כמו חולץ פקקים או פותחן קופסאות יכול להיות יעיל ואפקטיבי יותר למשימה ספציפית. באופן דומה, טכניקות עיבוד שפה טבעית מסורתיות, כמו אשכול מסמכים, זיהוי נושאים, וסיווג, יכולות לעתים קרובות לספק פתרונות ממוקדים וחסכוניים יותר עבור היבטים מסוימים של צינור עיבוד השפה הטבעית שלך.

אחד היתרונות המרכזיים של טכניקות עיבוד שפה טבעית מסורתיות הוא היעילות החישובית שלהן. שיטות אלה, המתבססות לעתים קרובות על מודלים סטטיסטיים פשוטים יותר או גישות מבוססות חוקים, יכולות לעבד כמויות גדולות של נתוני טקסט מהר יותר ועם תקורה חישובית נמוכה יותר בהשוואה למודלי שפה גדולים. זה הופך אותן למתאימות במיוחד למשימות הכרוכות בניתוח וארגון של אוספי מסמכים גדולים, כמו אשכול מאמרים דומים או זיהוי נושאים מרכזיים בתוך אוסף של טקסטים.

יתר על כן, טכניקות מסורתיות של עיבוד שפה טבעית יכולות להשיג לעתים קרובות דיוק ורמת דיוק גבוהים עבור משימות ספציפיות, במיוחד כאשר הן מאומנות על מאגרי נתונים ייעודיים לתחום מסוים. לדוגמה, מסווג מסמכים מכוונן היטב המשתמש באלגוריתמי למידת

מכונה מסורתיים כמו מכונות וקטורים תומכים SVM)) או נאיבי בייס יכול לסווג מסמכים לקטגוריות מוגדרות מראש בדיוק רב ובעלות חישובית מינימלית.

עם זאת, מודלים שפתיים גדולים באמת מצטיינים כשמדובר במשימות הדורשות הבנה עמוקה יותר של שפה, הקשר וחשיבה. יכולתם לייצר טקסט קוהרנטי ורלוונטי להקשר, לענות על שאלות ולסכם קטעים ארוכים אינה ניתנת להשוואה לשיטות עיבוד שפה טבעית מסורתיות. מודלים שפתיים גדולים יכולים להתמודד ביעילות עם תופעות לשוניות מורכבות, כגון דו-משמעות, התייחסות הדדית וביטויים אידיומטיים, מה שהופך אותם לחיוניים למשימות הדורשות יצירת שפה טבעית או הבנה.

הכוח האמיתי טמון בשילוב טכניקות עיבוד שפה טבעית מסורתיות עם מודלים שפתיים גדולים ליצירת גישות היברידיות המנצלות את החוזקות של שניהם. באמצעות שימוש בשיטות עיבוד שפה טבעית מסורתיות למשימות כמו קדם-עיבוד מסמכים, אשכול וחילוץ נושאים, ניתן לארגן ולמבנה את נתוני הטקסט שלך ביעילות. מידע מובנה זה יכול אז להיות מוזן למודלים שפתיים גדולים למשימות מתקדמות יותר, כגון יצירת תקצירים, מענה על שאלות או יצירת דוחות מקיפים.

לדוגמה, בואו נתבונן במקרה שימוש שבו אתם רוצים לייצר דוח מגמות עבור תחום ספציפי המבוסס על קורפוס גדול של מסמכי מגמות בודדים. במקום להסתמך אך ורק על מודלים שפתיים גדולים, שיכולים להיות יקרים מבחינה חישובית ולצרוך זמן רב לעיבוד כמויות גדולות של טקסט, ניתן להשתמש בגישה היברידית:

1. השתמשו בטכניקות עיבוד שפה טבעית מסורתיות, כגון מידול נושאים (למשל, הקצאת דיריכלה חבויה) או אלגוריתמי אשכול (למשל, K-means), כדי לקבץ יחד מסמכי מגמות דומים ולזהות נושאים ותמות מרכזיות בתוך הקורפוס.
2. הזינו את המסמכים המקובצים והנושאים שזוהו למודל שפתי גדול, תוך ניצול יכולות הבנת השפה והיצירה העדיפות שלו ליצירת תקצירים קוהרנטיים ואינפורמטיביים עבור כל אשכול או נושא.
3. לבסוף, השתמשו במודל השפתי הגדול כדי לייצר דוח מגמות מקיף על ידי שילוב התקצירים הבודדים, הדגשת המגמות המשמעותיות ביותר, ומתן תובנות והמלצות המבוססות על המידע המצטבר.

על ידי שילוב טכניקות עיבוד שפה טבעית מסורתיות עם מודלים שפתיים גדולים בדרך זו, ניתן לעבד ביעילות כמויות גדולות של נתוני טקסט, לחלץ תובנות משמעותיות, ולייצר דוחות באיכות גבוהה תוך אופטימיזציה של משאבים חישוביים ועלויות.

כאשר אתם יוצאים לדרך עם פרויקטי עיבוד שפה טבעית (NLP), חיוני להעריך בקפידה את הדרישות והאילוצים הספציפיים של כל משימה ולשקול כיצד ניתן לנצל יחד שיטות מסורתיות של עיבוד שפה טבעית ומודלים שפתיים גדולים להשגת התוצאות הטובות ביותר. באמצעות שילוב היעילות והדיוק של הטכניקות המסורתיות עם הגמישות והעוצמה של המודלים השפתיים הגדולים, תוכלו ליצור פתרונות עיבוד שפה טבעית יעילים וחסכוניים המספקים ערך למשתמשים ולבעלי העניין שלכם.

שימוש בכלים

בתחום פיתוח היישומים מבוססי הבינה המלאכותית, המושג "שימוש בכלים" או "קריאה לפונקציות" התפתח כטכניקה רבת עוצמה המאפשרת ל-LLM שלך להתחבר לכלים חיצוניים, ממשקי API, פונקציות, מסדי נתונים ומשאבים אחרים. גישה זו מאפשרת מגוון התנהגויות עשיר יותר מאשר רק פלט טקסט, ואינטראקציות דינמיות יותר בין רכיבי הבינה המלאכותית לבין שאר המערכת האקולוגית של היישום שלך. כפי שנבחן בפרק זה, שימוש בכלים גם מעניק לך את האפשרות לגרום למודל הבינה המלאכותית שלך לייצר נתונים בצורה מובנית.

מהו שימוש בכלים?

שימוש בכלים, המוכר גם כקריאה לפונקציות, הוא טכניקה המאפשרת למפתחים להגדיר רשימה של פונקציות שאיתן ה-LLM יכול לתקשר במהלך תהליך היצירה. כלים אלה יכולים לנוע מפונקציות שירות פשוטות ועד לממשקי API מורכבים או שאילתות מסד נתונים. על ידי מתן גישה ל-LLM לכלים אלה, מפתחים יכולים להרחיב את יכולות המודל ולאפשר לו לבצע משימות הדורשות ידע או פעולות חיצוניות.

איור 8.8. דוגמה להגדרת פונקציה עבור עובד בינה מלאכותית המנתח מסמכים

```
FUNCTION = {
  name: "save_analysis",
  description: "document for data analysis Save",
  parameters: {
    type: "object",
    properties: {
      title: {
        type: "string",
        maxLength: 140
      },
      summary: {
        type: "string",
        description: "with summary multi-paragraph comprehensive
applicable) (if sections of list and overview                    "
      },
      tags: {
        type: "array",
        items: {
          type: "string",
          description: "themes main representing tags lowercase
document the of                    "
        }
      }
    },
    "required": %w[tags summary title]
  }
}.freeze
```

הרעיון המרכזי מאחורי השימוש בכלים הוא לתת ל-LLM את היכולת לבחור ולהפעיל באופן

דינמי את הכלים המתאימים בהתבסס על קלט המשתמש או המשימה הנדרשת. במקום להסתמך אך ורק על הידע המוטמע מראש במודל, השימוש בכלים מאפשר ל-LLM לנצל משאבים חיצוניים כדי לייצר תשובות מדויקות, רלוונטיות ושימושיות יותר. השימוש בכלים הופך טכניקות כמו RAG (ייצור מועשר באחזור) לקלות הרבה יותר ליישום מכפי שהיו אחרת.

שימו לב שאלא אם צוין אחרת, ספר זה מניח שמודל הבינה המלאכותית שלכם אינו מחובר לכלים מובנים בצד השרת. כל כלי שתרצו להנגיש לבינה המלאכותית שלכם חייב להיות מוצהר על ידכם באופן מפורש בכל בקשת API, עם הוראות לביצוע הפעלתו אם וכאשר הבינה המלאכותית שלכם תודיע לכם שהיא מעוניינת להשתמש בכלי זה בתשובתה.

הפוטנציאל של השימוש בכלים

השימוש בכלים פותח מגוון רחב של אפשרויות ליישומי בינה מלאכותית. הנה מספר דוגמאות למה שניתן להשיג באמצעות שימוש בכלים:

1. **צ'אטבוטים ועוזרים וירטואליים**: על ידי חיבור LLM לכלים חיצוניים, צ'אטבוטים ועוזרים וירטואליים יכולים לבצע משימות מורכבות יותר, כמו אחזור מידע ממסדי נתונים, ביצוע קריאות API, או אינטראקציה עם מערכות אחרות. לדוגמה, צ'אטבוט יכול להשתמש בכלי CRM כדי לשנות את הסטטוס של עסקה בהתאם לבקשת המשתמש.
2. **ניתוח נתונים ותובנות**: ניתן לחבר LLM לכלי ניתוח נתונים או ספריות כדי לבצע משימות עיבוד נתונים מתקדמות. זה מאפשר ליישומים להפיק תובנות, לבצע ניתוחים השוואתיים, או לספק המלצות מבוססות נתונים בהתאם לשאילתות המשתמש.
3. **חיפוש ואחזור מידע**: השימוש בכלים מאפשר ל-LLM לתקשר עם מנועי חיפוש, מסדי נתונים וקטוריים, או מערכות אחזור מידע אחרות. על ידי המרת שאילתות משתמש לשאילתות חיפוש, ה-LLM יכול לאחזר מידע רלוונטי ממקורות מרובים ולספק תשובות מקיפות לשאלות המשתמש.
4. **אינטגרציה עם שירותים חיצוניים**: השימוש בכלים מאפשר אינטגרציה חלקה בין יישומי בינה מלאכותית לבין שירותים חיצוניים או ממשקי API. לדוגמה, LLM יכול לתקשר

עם API של מזג אוויר כדי לספק עדכוני מזג אוויר בזמן אמת או עם API של תרגום כדי לייצר תשובות במספר שפות.

תהליך העבודה עם כלים

זרימת העבודה של שימוש בכלים כוללת בדרך כלל ארבעה שלבים עיקריים:

1. הכללת הגדרות פונקציות בהקשר הבקשה שלך
2. בחירת כלים דינמית (או מפורשת)
3. הרצת פונקציה/ות
4. המשך אופציונלי של הפקודה המקורית

הבה נסקור כל אחד מהשלבים הללו בפירוט.

הכללת הגדרות פונקציות בהקשר הבקשה שלך

הבינה המלאכותית יודעת אילו כלים עומדים לרשותה מכיוון שאתה מספק לה רשימה כחלק מבקשת ההשלמה שלך (בדרך כלל מוגדרת כפונקציות באמצעות גרסה של סכמת).

תחביר ההגדרה המדויק של הכלים תלוי בדגם הספציפי.

כך מגדירים פונקציית weather_get ב-Claude 3:

```
{
   "name": "get_weather",
   "description": location" given a in weather current the "Get,
   "input_schema": {
      "type": "object",
      "properties": {
         "location": {
            "type": "string",
            "description": CA" ,Francisco San e.g. ,state and city "The
         },
         "unit": {
            "type": "string",
            "enum": ["celsius", "fahrenheit"],
            "description": temperature" of unit "The
      }
   },
   "required": ["location"]
   }
}
```

וכך תגדיר את אותה הפונקציה עבור GPT-4, כאשר אתה מעביר אותה כערך לפרמטר tools:

```
{
   "name": "get_current_weather",
   "description": location" given a in weather current the "Get,
   "parameters": {
      "type": "object",
      "properties": {
         "location": {
            "type": "string",
            "description": CA" ,Francisco San e.g. ,state and city "The,
         },
         "unit": {
            "type": "string",
            "enum": ["celsius", "fahrenheit"],
            "description": temperature" of unit "The
         },
      },
      "required": ["location"],
   },
}
```

A> כמעט זהה, רק שונה ללא סיבה נראית לעין! כמה מעצבן.

הגדרות פונקציה מפרטות שם, תיאור ופרמטרי קלט. ניתן להגדיר פרמטרי קלט נוספים באמצעות מאפיינים כמו מספרי מנייה להגבלת הערכים המקובלים, וציון האם פרמטר הוא נדרש או לא.

בנוסף להגדרות הפונקציה עצמן, ניתן לכלול גם הוראות או הקשר לגבי מדוע וכיצד להשתמש בפונקציה בהנחיית המערכת.

לדוגמה, כלי חיפוש האינטרנט שלי ב-Olympia כולל את הנחיית המערכת הזו, אשר מזכירה לבינה המלאכותית שהכלים המוזכרים עומדים לרשותה:

```
The ‘google_search‘ and ‘realtime_search‘ functions let you do research
on behalf of the user. In contrast to Google, realtime search is powered
by Perplexity and provides real-time information to curated current events
databases and news sources. Make sure to include URLs in your response so
user can do followup research.
```

מתן תיאורים מפורטים נחשב לגורם החשוב ביותר בביצועי הכלי. התיאורים שלך צריכים להסביר כל פרט אודות הכלי, כולל:

- מה הכלי עושה
- מתי יש להשתמש בו (ומתי לא)
- מה המשמעות של כל פרמטר וכיצד הוא משפיע על התנהגות הכלי
- כל הסתייגות או מגבלה חשובה החלה על מימוש הכלי

ככל שתוכל לספק יותר הקשר לבינה המלאכותית אודות הכלים שלך, כך היא תהיה טובה יותר בהחלטה מתי וכיצד להשתמש בהם. לדוגמה, Anthropic ממליצה על לפחות 3-4 משפטים לתיאור כל כלי עבור סדרת Claude 3 שלה, ויותר אם הכלי מורכב.

זה לא בהכרח אינטואיטיבי, אבל תיאורים נחשבים חשובים יותר מדוגמאות. בעוד שאתה יכול לכלול דוגמאות לשימוש בכלי בתיאור שלו או בהנחיה המצורפת, זה פחות חשוב מאשר הסבר ברור ומקיף של מטרת הכלי והפרמטרים שלו. הוסף דוגמאות רק לאחר שפיתחת באופן מלא את התיאור.

הנה דוגמה למפרט פונקציית API בסגנון Stripe:

```
{
  "name": "createPayment",
  "description": request" payment new a "Create,
  "parameters": {
    "type": "object",
    "properties": {
      "transaction_amount": {
        "type": "number",
        "description": paid" be to amount "The
      },
      "description": {
        "type": "string",
        "description": payment" the of description brief "A
      },
      "payment_method_id": {
        "type": "string",
        "description": used" be to method payment "The
      },
      "payer": {
        "type": "object",
        "description": ,name their including ,payer the about "Information
number" identification and ,email                    ,
        "properties": {
          "name": {
            "type": "string",
            "description": name" payer's "The
          },
          "email": {
            "type": "string",
            "description": address" email payer's "The
          },
          "identification": {
            "type": "object",
            "description": number" identification payer's "The,
            "properties": {
              "type": {
                "type": "string",
                "description": CNPJ)" ,CPF (e.g. document "Identification
              },
              "number": {
                "type": "string",
                "description": number" identification "The
```

```
            }
          },
          "required": [ "type", "number" ]
        }
      },
      "required": [ "name", "email", "identification" ]
    }
  }
}
```

למעשה, לחלק מהמודלים יש קושי בטיפול במפרטי פונקציות מקוננות ובטיפול בטיפוסי נתונים מורכבים כמו מערכים, מילונים וכו'. אבל תיאורטית, אתה אמור להיות מסוגל לספק מפרטי סכמת JSON בעומק שרירותי!

בחירת כלים דינמית

כאשר אתה מבצע השלמת צ'אט הכוללת הגדרות כלים, ה-LLM בוחר באופן דינמי את הכלי/ם המתאים/ים ביותר לשימוש ומייצר את פרמטרי הקלט הנדרשים לכל כלי.

בפועל, היכולת של הבינה המלאכותית לקרוא בדיוק לפונקציה הנכונה, ולעקוב בדיוק אחר המפרט שלך עבור הקלט היא לא תמיד מדויקת. הורדת פרמטר הטמפרטורה עד ל-0.0 עוזרת מאוד, אבל מניסיוני עדיין תראה שגיאות מדי פעם. כשלונות אלה כוללים שמות פונקציות הזויים, פרמטרי קלט שגויים או פשוט חסרים. הפרמטרים מועברים כ-JSON, מה שאומר שלפעמים תראה שגיאות שנגרמות בגלל JSON קטוע, מצוטט לא נכון, או שבור בדרך אחרת.

דפוסי ריפוי נתונים עצמי יכולים לעזור לתקן אוטומטית קריאות פונקציה שנשברות בגלל שגיאות תחביר.

בחירת כלים מאולצת (או מפורשת)

חלק מהמודלים נותנים לך את האפשרות לאלץ קריאה לפונקציה מסוימת, כפרמטר בבקשה. אחרת, ההחלטה אם לקרוא לפונקציה או לא נתונה לחלוטין לשיקול דעתה של הבינה המלאכותית.

היכולת לאלץ קריאת פונקציה היא קריטית בתרחישים מסוימים בהם אתה רוצה להבטיח שכלי או פונקציה ספציפיים יבוצעו, ללא קשר לתהליך הבחירה הדינמי של הבינה המלאכותית. ישנן מספר סיבות מדוע יכולת זו חשובה:

1. **שליטה מפורשת**: ייתכן שאתה משתמש בבינה המלאכותית כרכיב בדיד או בתהליך עבודה מוגדר מראש המחייב הפעלה של פונקציה מסוימת בזמן מסוים. על ידי אילוץ הקריאה, אתה יכול להבטיח שהפונקציה הרצויה תיקרא במקום לבקש יפה מהבינה המלאכותית לעשות זאת.
2. **דיבוג ובדיקות**: בעת פיתוח ובדיקה של יישומים מבוססי בינה מלאכותית, היכולת לאלץ קריאות פונקציה היא בעלת ערך רב למטרות דיבוג. על ידי הפעלה מפורשת של פונקציות ספציפיות, אתה יכול לבודד ולבדוק רכיבים בודדים של היישום שלך. זה מאפשר לך לאמת את נכונות מימושי הפונקציות, לתקף את פרמטרי הקלט, ולהבטיח שמתקבלות התוצאות הצפויות.
3. **טיפול במקרי קצה**: ייתכנו מקרי קצה או תרחישים חריגים שבהם תהליך הבחירה הדינמי של הבינה המלאכותית עשוי שלא לבחור להפעיל פונקציה שהיא אמורה להפעיל, ואתם יודעים זאת בהתבסס על תהליכים חיצוניים. במקרים כאלה, היכולת לכפות קריאה לפונקציה מאפשרת לכם לטפל במצבים אלה באופן מפורש. הגדירו כללים או תנאים בלוגיקת היישום שלכם כדי לקבוע מתי לעקוף את שיקול הדעת של הבינה המלאכותית.
4. **עקביות ושחזור**: אם יש לכם רצף ספציפי של פונקציות שצריכות להיות מופעלות בסדר מסוים, כפיית הקריאות מבטיחה שאותו רצף יישמר בכל פעם. זה חשוב במיוחד ביישומים שבהם עקביות והתנהגות צפויה הן קריטיות, כמו במערכות פיננסיות או בסימולציות מדעיות.
5. **אופטימיזציית ביצועים**: במקרים מסוימים, כפיית קריאה לפונקציה יכולה להוביל לאופטימיזציית ביצועים. אם אתם יודעים שפונקציה מסוימת נדרשת למשימה מסוימת

ושתהליך הבחירה הדינמי של הבינה המלאכותית עלול להוסיף תקורה מיותרת, אתם יכולים לעקוף את תהליך הבחירה ולהפעיל ישירות את הפונקציה הנדרשת. זה יכול לעזור להפחית את זמן התגובה ולשפר את היעילות הכוללת של היישום שלכם.

לסיכום, היכולת לכפות קריאות לפונקציות ביישומים מבוססי בינה מלאכותית מספקת שליטה מפורשת, מסייעת בניפוי באגים ובדיקות, מטפלת במקרי קצה, ומבטיחה עקביות ויכולת שחזור. זהו כלי חזק בארסנל שלכם, אבל עלינו לדון בעוד היבט אחד של תכונה חשובה זו.

במקרים רבים של קבלת החלטות, אנחנו תמיד רוצים שהמודל יבצע קריאה לפונקציה ויתכן שלעולם לא נרצה שהמודל יגיב רק עם הידע הפנימי שלו. לדוגמה, אם אתם מנתבים בין מודלים מרובים המתמחים במשימות שונות (קלט רב-לשוני, מתמטיקה וכו'), ייתכן שתשתמשו במודל קריאת הפונקציות כדי להעביר בקשות לאחד ממודלי העזר ולעולם לא להגיב באופן עצמאי.

פרמטר בחירת כלים

ומודלי שפה אחרים התומכים בקריאת פונקציות מספקים לכם פרמטר choice_tool לשליטה באם נדרש שימוש בכלי כחלק מההשלמה. לפרמטר זה יש שלושה ערכים אפשריים: GPT-4

- auto נותן לבינה המלאכותית שיקול דעת מלא לגבי השימוש בכלי או פשוט להגיב
- required אומר לבינה המלאכותית שהיא חייבת לקרוא לכלי כלשהו להגיב, אך משאיר את בחירת הכלי לבינה המלאכותית
- האפשרות השלישית היא להגדיר את הפרמטר של function_of_name שאתם רוצים לכפות. עוד על כך בסעיף הבא.

שים לב שאם תגדיר את בחירת הכלי כ-required, המודל יהיה מוכרח לבחור את הפונקציה הרלוונטית ביותר לקריאה מתוך אלו שסופקו לו, גם אם אף אחת מהן לא באמת מתאימה לבקשה. נכון למועד הפרסום, אינני מכיר מודל שיחזיר תגובת calls_tool ריקה, או ישתמש בדרך אחרת להודיע לך שלא מצא פונקציה מתאימה לקריאה.

כפיית פונקציה לקבלת פלט מובנה

היכולת לכפות קריאת פונקציה מעניקה לך דרך לכפות מידע מובנה מהשלמת צ'אט במקום להצטרך לחלץ אותו בעצמך מתוך תגובת טקסט רגיל.

> מדוע כפיית פונקציות לקבלת פלט מובנה היא עניין גדול? פשוט מאוד, כי חילוץ מידע מובנה מפלט של מודל שפה גדול הוא כאב ראש. אתה יכול להקל על חייך מעט על ידי בקשת נתונים ב-XML, אבל אז אתה צריך לנתח .XML ומה אתה עושה כשה-XML הזה חסר כי הבינה המלאכותית שלך ענתה: "מצטער, אבל איני יכול ליצור את הנתונים שביקשת כי בלה, בלה, בלה..."

כאשר משתמשים בכלים בדרך זו:

- כדאי כנראה להגדיר כלי יחיד בבקשה שלך
- זכור לכפות שימוש בפונקציה שלו באמצעות הפרמטר choice_tool
- זכור שהמודל יעביר את הקלט לכלי, לכן שם הכלי והתיאור צריכים להיות מנקודת המבט של המודל, לא שלך

הנקודה האחרונה ראויה לדוגמה להבהרה. נניח שאתה מבקש מהבינה המלאכותית לבצע ניתוח רגשות על טקסט של משתמש. שם הפונקציה לא יהיה sentiment_analyze, אלא משהו כמו analysis_sentiment_save. הבינה המלאכותית היא זו שמבצעת את ניתוח הרגשות, לא הכלי. כל מה שהכלי עושה (מנקודת המבט של הבינה המלאכותית) הוא לשמור את תוצאות הניתוח.

הנה דוגמה לשימוש ב-Claude 3 כדי לתעד סיכום של תמונה ב-JSON מובנה היטב, הפעם משורת הפקודה באמצעות curl:

```
curl https://api.anthropic.com/v1/messages \
    --header application/json" "content-type: \
    --header " x-api-key:$ANTHROPIC_API_KEY" \
    --header 2023-06-01" "anthropic-version: \
    --header tools-2024-04-04" "anthropic-beta: \
    --data \
'{
,"claude-3-sonnet-20240229" "model":
,1024 "max_tokens":
[{ "tools":
,"record_summary" "name":
,JSON." well-structured into image of summary "Record "description":
{ "input_schema":
,"object" "type":
{ "properties":
{ "key_colors":
,"array" "type":
{ "items":
,"object" "type":
{ "properties":
{ "r":
,"number" "type":
1.0]" ,[0.0 value "red "description":
,}
{ "g":
,"number" "type":
1.0]" ,[0.0 value "green "description":
,}
{ "b":
,"number" "type":
1.0]" ,[0.0 value "blue "description":
,}
{ "name":
,"string" "type":
name color "Human-readable "description":
e.g. ,snake_case in
\"olive_green\"or
\"turquoise\""
}
,}
] "name" ,"b" ,"g" ,"r" [ "required":
,}
```

```
less." or Four image. the in colors "Key "description":
,}
{ "description":
,"string" "type":
max." sentences 1-2 description. "Image "description":
,}
{ "estimated_year":
,"integer" "type":
,taken was image the that year "Estimated "description":
the if this set Only photo. a it is if
non-fictional. be to appears image
okay!" are estimates Rough
}
,}
] "description" ,"key_colors" [ "required":
}
,}]
[ "messages":
{
,"user" "role":
[ "content":
{
,"image" "type":
{ "source":
,"base64" "type":
"' "media_type":          $IMAGE_MEDIA_TYPE,'"
"' "data":          $IMAGE_BASE64'"
}
,}
{
,"text" "type":
image." this describe to ‘‘record_summary "Use "text":
}
]
}
]
}'
```

בדוגמה המוצגת, אנחנו משתמשים במודל Claude 3 מבית Anthropic כדי ליצור סיכום JSON מובנה של תמונה. כך זה עובד:

1. אנחנו מגדירים כלי בודד בשם summary_record במערך ה-tools של ה-payload בבקשה. כלי זה אחראי לתיעוד סיכום התמונה ב-JSON מובנה היטב.
2. לכלי summary_record יש schema_input המגדיר את המבנה הצפוי של פלט ה-JSON. הוא מגדיר שלושה מאפיינים:

- colors_key: מערך של אובייקטים המייצגים את הצבעים המרכזיים בתמונה. לכל אובייקט צבע יש מאפיינים עבור ערכי אדום, ירוק וכחול (בטווח שבין 0.0 ל-1.0) ושם צבע קריא לאדם בפורמט case._snake
- description: מאפיין מחרוזת המיועד לתיאור קצר של התמונה, מוגבל ל-1-2 משפטים.
- year_estimated: מאפיין מספר שלם אופציונלי עבור השנה המשוערת בה צולמה התמונה, אם נראה שמדובר בצילום לא בדיוני.

3. במערך ה-messages, אנחנו מספקים את נתוני התמונה כמחרוזת מקודדת ב-base64 יחד עם סוג המדיה. זה מאפשר למודל לעבד את התמונה כחלק מהקלט.
4. אנחנו גם מנחים את Claude להשתמש בכלי summary_record כדי לתאר את התמונה.
5. כאשר הבקשה נשלחת למודל Claude 3, הוא מנתח את התמונה ומייצר סיכום JSON בהתאם ל-schema_input שהוגדר. המודל מחלץ את הצבעים המרכזיים, מספק תיאור קצר, ומעריך את שנת צילום התמונה (אם רלוונטי).
6. סיכום ה-JSON שנוצר מועבר כפרמטרים לכלי summary_record, ומספק ייצוג מובנה של המאפיינים המרכזיים של התמונה.

באמצעות שימוש בכלי summary_record עם schema_input מוגדר היטב, אנחנו יכולים לקבל סיכום JSON מובנה של תמונה מבלי להסתמך על חילוץ טקסט פשוט. גישה זו מבטיחה שהפלט יהיה בפורמט עקבי וניתן יהיה לנתח ולעבד אותו בקלות על ידי רכיבי ההמשך של האפליקציה.

היכולת לכפות קריאה לפונקציה ולציין את מבנה הפלט הצפוי היא תכונה חזקה של השימוש בכלים ביישומים מבוססי בינה מלאכותית. היא מאפשרת למפתחים לקבל יותר שליטה על הפלט המיוצר ומפשטת את השילוב של מידע שנוצר על ידי בינה מלאכותית בזרימת העבודה של האפליקציה שלהם.

הרצת פונקציה/ות

הגדרת פונקציות, והפעלת את הבינה המלאכותית שלך, אשר החליטה שעליה לקרוא לאחת הפונקציות שלך. כעת הגיע הזמן שקוד היישום שלך או הספרייה, אם אתה משתמש בג'ם Ruby כמו raix-rails, ישגרו את קריאת הפונקציה ואת הפרמטרים שלה למימוש המתאים בקוד היישום שלך.

קוד היישום שלך מחליט מה לעשות עם תוצאות הרצת הפונקציה. יתכן שמדובר בשורת קוד בודדת ב-lambda, או אולי זה כולל קריאה ל-API חיצוני. יתכן שזה כולל קריאה לרכיב בינה מלאכותית אחר, או אולי זה כולל מאות ואפילו אלפי שורות קוד בשאר המערכת שלך. זה לחלוטין תלוי בך.

לפעמים קריאת הפונקציה היא סוף הפעולה, אבל אם התוצאות מייצגות מידע בשרשרת חשיבה שאמורה להימשך על ידי הבינה המלאכותית, אז קוד היישום שלך צריך להכניס את תוצאות ההרצה לתמליל הצ'אט ולאפשר לבינה המלאכותית להמשיך בעיבוד.

לדוגמה, הנה הצהרת פונקציה של Raix המשמשת את AccountManager של Olympia לתקשורת עם הלקוחות שלנו כחלק מתזמור תהליכי עבודה חכמים עבור שירות לקוחות.

```
1  class AccountManager
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    functions... other of lots #
6
7    function :notify_account_owner,
8      "changed subscription if dollars Mention UUID. share Don't",
9      message: { type: "string" } do |arguments|
10       account.owner.freeform_notify(
11         subject: "Notification Change Account",
12         message: arguments[:message]
13       )
14       "owner account Notified"
15     end
```

ייתכן שלא ברור מיד מה קורה כאן, אז אפרק זאת.

1. המחלקה AccountManager מגדירה פונקציות רבות הקשורות לניהול חשבון. היא יכולה לשנות את התוכנית שלך, להוסיף ולהסיר חברי צוות, בין היתר.
2. ההוראות ברמה העליונה שלה אומרות ל-AccountManager שעליו להודיע לבעל החשבון על תוצאות בקשת שינוי החשבון, באמצעות הפונקציה owner_account_notify.
3. ההגדרה התמציתית של הפונקציה כוללת את:

- שמה
- תיאורה
- הפרמטרים שלה :message { type: "string" }
- בלוק לביצוע כאשר הפונקציה נקראת

לאחר עדכון התמליל עם תוצאות בלוק הפונקציה, מתבצעת קריאה נוספת למתודה chat_-completion. מתודה זו אחראית לשליחת תמליל השיחה המעודכן בחזרה למודל ה-AI לעיבוד נוסף. אנו מתייחסים לתהליך זה כלולאת שיחה.

כאשר מודל ה-AI מקבל בקשת השלמת צ'אט חדשה עם תמליל מעודכן, יש לו גישה לתוצאות הפונקציה שבוצעה קודם לכן. הוא יכול לנתח תוצאות אלה, לשלב אותן בתהליך קבלת ההחלטות שלו, ולייצר את התגובה או הפעולה הבאה בהתבסס על ההקשר המצטבר של השיחה. הוא יכול לבחור לבצע פונקציות נוספות בהתבסס על ההקשר המעודכן, או לייצר תגובה סופית לבקשה המקורית אם הוא קובע שאין צורך בקריאות פונקציה נוספות.

המשך אופציונלי של הבקשה המקורית

כאשר אתה שולח את תוצאות הכלי בחזרה ל-LLM וממשיך בעיבוד הבקשה המקורית, ה-AI משתמש בתוצאות אלה כדי לקרוא לפונקציות נוספות או לייצר תגובת טקסט פשוטה סופית.

מודלים מסוימים כמו Command-R של Cohere יכולים לצטט את הכלים הספציפיים בהם השתמשו בתשובותיהם, מה שמספק שקיפות ויכולת מעקב נוספות.

בהתאם למודל בשימוש, תוצאות קריאת הפונקציה יהיו בהודעות תמליל שיש להן תפקיד מיוחד משלהן או ישתקפו בתחביר אחר. אבל החלק החשוב הוא שהנתונים האלה יהיו בתמליל, כך שה-AI יוכל להתחשב בהם בעת שהוא מחליט מה לעשות בהמשך.

שגיאה נפוצה (ופוטנציאלית יקרה) היא לשכוח להוסיף את תוצאות הפונקציה לתמליל לפני שממשיכים בצ'אט. כתוצאה מכך, ה-AI יקבל בקשה באופן שזהה בעיקרון לאופן שבו הוא קיבל אותה לפני שקרא לפונקציה בפעם הראשונה. במילים אחרות, מבחינת ה-AI, הוא עדיין לא קרא לפונקציה. אז הוא קורא לה שוב. ושוב. ושוב, לנצח עד שתפסיק אותו. נקווה שההקשר שלך לא היה גדול מדי, והמודל שלך לא היה יקר מדי!

שיטות עבודה מומלצות לשימוש בכלים

כדי להפיק את המרב משימוש בכלים, שקלו את השיטות המומלצות הבאות.

הגדרות תיאוריות

ספקו שמות ותיאורים ברורים ומפורטים עבור כל כלי ופרמטרי הקלט שלו. זה עוזר ל-LLM להבין טוב יותר את המטרה והיכולות של כל כלי.

> מניסיוני האישי אני יכול לומר שהחוכמה המקובלת ש"מתן שמות זה קשה" תקפה גם כאן; ראיתי תוצאות שונות לחלוטין מ-LLMs רק בגלל שינוי שמות של פונקציות או ניסוח של תיאורים. לפעמים הסרת תיאורים אפילו משפרת את הביצועים.

עיבוד תוצאות הכלים

בעת העברת תוצאות כלים בחזרה ל-LLM, ודאו שהן מובנות היטב ומקיפות. השתמשו במפתחות וערכים משמעותיים כדי לייצג את הפלט של כל כלי. התנסו בפורמטים שונים וראו מה עובד הכי טוב, החל מ-JSON ועד טקסט רגיל.

מפרש התוצאות מתמודד עם אתגר זה על ידי שימוש ב-AI לניתוח התוצאות ומתן הסברים ידידותיים למשתמש, סיכומים, או תובנות עיקריות.

טיפול בשגיאות

יישמו מנגנוני טיפול בשגיאות חזקים כדי לטפל במקרים בהם ה-LLM עשוי לייצר פרמטרי קלט לא תקינים או לא נתמכים עבור קריאות לכלים. טפלו והתאוששו בצורה חלקה מכל שגיאה שעלולה להתרחש במהלך הפעלת הכלי.

תכונה נהדרת במיוחד של ה-AI היא שהוא מבין הודעות שגיאה! מה שאומר שאם אתם עובדים בגישה מהירה ופשוטה, אתם יכולים פשוט לתפוס כל חריגה שנוצרת ביישום של כלי, ולהעביר אותה בחזרה ל-AI כדי שידע מה קרה!

לדוגמה, הנה גרסה מצומצמת של היישום של חיפוש גוגל ב-Olympia:

```
1  def google_search(conversation, params)
2    conversation.update_cstatus("Google... Searching")
3    query = params[:query]
4    search = GoogleSearch.new(query).get_hash
5
6    conversation.update_cstatus("results... Summarizing")
7    SummarizeKnowledgeGraph.new.perform(conversation, search.to_json)
8  rescue StandardError => e
9    Honeybadger.notify(e)
10   { error: e.message }.inspect
11 end
```

חיפושי Google ב-Olympia הם תהליך דו-שלבי. קודם מבצעים את החיפוש, ואז מסכמים את התוצאות. אם יש כישלון, לא משנה מה הוא, הודעת השגיאה נארזת ונשלחת בחזרה ל-AI. טכניקה זו היא הבסיס לכמעט כל תבניות טיפול חכם בשגיאות

לדוגמה, נניח שקריאת ה-GoogleSearch API נכשלת עקב שגיאת 503 Service Unavailable. השגיאה מועברת למעלה לטיפול השגיאות העליון, ותיאור השגיאה נשלח בחזרה ל-AI כתוצאה של קריאת הפונקציה. במקום פשוט להציג למשתמש מסך ריק או שגיאה טכנית, ה-AI אומר משהו כמו "אני מצטער, אבל איני יכול לגשת ליכולות חיפוש Google שלי כרגע. אני יכול לנסות שוב מאוחר יותר, אם תרצה."

זה עשוי להיראות כמו תרגיל חכם בלבד, אבל שקלו סוג אחר של שגיאה, כזה שבו ה-AI קורא ל-API חיצוני ויש לו שליטה ישירה על הפרמטרים שמועברים ל-API. אולי הוא טעה באופן שבו

הוא יצר את הפרמטרים האלה? בהנחה שהודעת השגיאה מה-API החיצוני מפורטת מספיק, העברת הודעת השגיאה בחזרה ל-AI הקורא מאפשרת לו לשקול מחדש את הפרמטרים האלה ולנסות שוב. באופן אוטומטי. לא משנה מה הייתה השגיאה.

עכשיו חשבו מה היה נדרש כדי לשחזר את סוג הטיפול החסין הזה בשגיאות בקוד רגיל. זה כמעט בלתי אפשרי.

שיפור איטרטיבי

אם ה-LLM אינו ממליץ על הכלים המתאימים או מייצר תגובות לא אופטימליות, יש לחזור ולשפר את הגדרות הכלים, התיאורים ופרמטרי הקלט. יש לשפר ולעדכן באופן מתמיד את הגדרות הכלים בהתבסס על ההתנהגות הנצפית והתוצאות הרצויות.

1. התחילו עם הגדרות כלים פשוטות: התחילו בהגדרת כלים עם שמות, תיאורים ופרמטרי קלט ברורים ותמציתיים. הימנעו מסיבוך יתר של הגדרות הכלים בהתחלה והתמקדו בפונקציונליות הבסיסית. לדוגמה, אם ברצונכם לשמור את תוצאות ניתוח הרגשות, התחילו עם הגדרה בסיסית כמו:

```
{
  "name": "save_sentiment_score",
  "description": score" sentiment generate and text user-provided "Analyze,
  "parameters": {
    "type": "object",
    "properties": {
      "score": {
        "type": "float",
        "description": (positive)" 1 to (negative) -1 from score "sentiment
      }
    },
    "required": ["score"]
  }
}
```

2. בדיקה ותצפית: לאחר שהגדרות הכלים הראשוניות במקומן, בדקו אותן עם הנחיות שונות וצפו כיצד מודל השפה הגדול מתקשר עם הכלי. שימו לב לאיכות ולרלוונטיות

של התגובות המופקות. אם המודל מייצר תגובות תת-אופטימליות, הגיע הזמן לטייב את הגדרות הכלים.

3. שיפור התיאורים: אם מודל השפה הגדול מבין לא נכון את מטרת הכלי, נסו לשפר את תיאור הכלי. ספקו יותר הקשר, דוגמאות, או הבהרות כדי להנחות את המודל בשימוש יעיל בכלי. למשל, תוכלו לעדכן את תיאור כלי ניתוח הרגשות כך שיתייחס באופן ספציפי יותר לטון הרגשי של קטע הטקסט המנותח:

```
{
  "name": "save_sentiment_score",
  "description": "Determine the overall emotional tone of a piece of text, such as reviews, social media posts, or feedback comments.",
  ...
}
```

4. התאמת פרמטרי קלט: אם ה-LLM מייצר פרמטרי קלט לא תקינים או לא רלוונטיים עבור כלי, שקול להתאים את הגדרות הפרמטרים. הוסף אילוצים ספציפיים יותר, כללי תיקוף, או דוגמאות כדי להבהיר את פורמט הקלט המצופה.
5. שיפור על בסיס משוב: עקוב באופן רציף אחר ביצועי הכלים שלך ואסוף משוב ממשתמשים או בעלי עניין. השתמש במשוב זה כדי לזהות תחומים הדורשים שיפור ובצע שיפורים חוזרים להגדרות הכלי. לדוגמה, אם משתמשים מדווחים שהניתוח אינו מטפל היטב בסרקזם, תוכל להוסיף הערה בתיאור:

```
{
  "name": "save_sentiment_score",
  "description": "Analyze the sentiment of a given text and return a score between -1 (negative) and 1 (positive). Note: Sarcasm should be considered negative.",
  ...
}
```

באמצעות שיפור חוזר של הגדרות הכלים שלך בהתבסס על התנהגות נצפית ומשוב, תוכל לשפר בהדרגה את הביצועים והיעילות של היישום מבוסס הבינה המלאכותית שלך. זכור לשמור על הגדרות הכלים ברורות, תמציתיות וממוקדות במשימה הספציפית. בדוק ואמת באופן קבוע את האינטראקציות של הכלים כדי להבטיח שהן מתואמות עם התוצאות הרצויות.

חיבור ושרשור כלים

אחד ההיבטים החזקים ביותר בשימוש בכלים, שרק נרמז עליו עד כה, הוא היכולת לחבר ולשרשר מספר כלים יחד כדי לבצע משימות מורכבות. באמצעות תכנון קפדני של הגדרות הכלים ופורמט הקלט/פלט שלהם, ניתן ליצור אבני בניין לשימוש חוזר שניתן לשלב בדרכים שונות.

הבה נבחן דוגמה שבה אתה בונה צינור ניתוח נתונים ליישום מבוסס הבינה המלאכותית שלך. ייתכן שיהיו לך הכלים הבאים:

1. **DataRetrieval:** כלי שמאחזר נתונים ממסד נתונים או API על בסיס קריטריונים מוגדרים.
2. **DataProcessing:** כלי שמבצע חישובים, טרנספורמציות או צבירות על הנתונים שאוחזרו.
3. **DataVisualization:** כלי שמציג את הנתונים המעובדים בפורמט ידידותי למשתמש, כגון תרשימים או גרפים.

על ידי שרשור כלים אלה יחד, ניתן ליצור זרימת עבודה חזקה שמאחזרת נתונים רלוונטיים, מעבדת אותם ומציגה את התוצאות בצורה משמעותית. כך עשויה להיראות זרימת העבודה של השימוש בכלים:

1. ה-LLM מקבל שאילתת משתמש המבקשת תובנות על נתוני מכירות עבור קטגוריית מוצרים מסוימת.
2. ה-LLM בוחר בכלי DataRetrieval ומייצר את פרמטרי הקלט המתאימים לאחזור נתוני המכירות הרלוונטיים ממסד הנתונים.
3. הנתונים שאוחזרו "מועברים" לכלי DataProcessing, אשר מחשב מדדים כמו הכנסה כוללת, מחיר מכירה ממוצע ושיעור צמיחה.
4. הנתונים המעובדים מועברים לאחר מכן לכלי DataVisualization, אשר יוצר תרשים או גרף מושך עין כדי לייצג את התובנות, ומעביר את כתובת ה-URL של התרשים בחזרה ל-LLM.

5. לבסוף, ה-LLM מייצר תשובה מעוצבת לשאילתת המשתמש באמצעות markdown, המשלבת את הנתונים המוחשיים ומספקת סיכום של הממצאים העיקריים.

באמצעות חיבור כלים אלה יחד, ניתן ליצור זרימת עבודה חלקה לניתוח נתונים שניתן לשלב בקלות ביישום שלך. היופי בגישה זו הוא שכל כלי יכול להיות מפותח ונבדק באופן עצמאי, ואז משולב בדרכים שונות לפתרון בעיות מגוונות.

כדי לאפשר הרכבה ושרשור חלקים של כלים, חשוב להגדיר פורמטים ברורים של קלט ופלט עבור כל כלי.

לדוגמה, הכלי DataRetrieval עשוי לקבל פרמטרים כמו פרטי התחברות למסד הנתונים, שם הטבלה ותנאי השאילתה, ולהחזיר את סט התוצאות כאובייקט JSON מובנה. הכלי DataProcessing יכול אז לצפות לקבל את אובייקט ה-JSON הזה כקלט ולייצר אובייקט JSON מעובד כפלט. על ידי סטנדרטיזציה של זרימת הנתונים בין הכלים, ניתן להבטיח תאימות ויכולת שימוש חוזר.

בעת תכנון המערכת האקולוגית של הכלים שלך, חשוב לחשוב על האופן שבו ניתן לשלב כלים שונים כדי לטפל במקרי שימוש נפוצים באפליקציה שלך. שקול ליצור כלים ברמה גבוהה המכמסים תהליכי עבודה או לוגיקה עסקית נפוצים, מה שיקל על ה-LLM לבחור ולהשתמש בהם ביעילות.

זכור, כוחו של השימוש בכלים טמון בגמישות ובמודולריות שהוא מספק. על ידי פירוק משימות מורכבות לכלים קטנים יותר הניתנים לשימוש חוזר, תוכל ליצור אפליקציה מונעת-AI חסינה ומסתגלת שיכולה להתמודד עם מגוון רחב של אתגרים.

כיוונים עתידיים

ככל שתחום פיתוח האפליקציות מונעות-AI מתפתח, אנו יכולים לצפות להתקדמות נוספת ביכולות השימוש בכלים. כמה כיוונים עתידיים אפשריים כוללים:

1. **שימוש מרובה-שלבים בכלים**: LLMs עשויים להיות מסוגלים להחליט כמה פעמים עליהם להשתמש בכלים כדי לייצר תגובה מספקת. זה עשוי לכלול סבבים מרובים של בחירת כלים והפעלתם בהתבסס על תוצאות ביניים.

2. **כלים מוגדרים מראש**: פלטפורמות AI עשויות לספק סט של כלים מוגדרים מראש שמפתחים יכולים לנצל באופן מיידי, כמו מפרשי Python, כלי חיפוש באינטרנט, או פונקציות שירות נפוצות.
3. **אינטגרציה חלקה**: ככל שהשימוש בכלים הופך נפוץ יותר, אנו יכולים לצפות לאינטגרציה טובה יותר בין פלטפורמות AI למסגרות פיתוח פופולריות, מה שיקל על מפתחים להטמיע שימוש בכלים באפליקציות שלהם.

שימוש בכלים הוא טכניקה חזקה המאפשרת למפתחים למנף את מלוא הפוטנציאל של LLMs באפליקציות מונעות-AI. על ידי חיבור LLMs לכלים ומשאבים חיצוניים, ניתן ליצור מערכות דינמיות, חכמות ומודעות להקשר יותר, שיכולות להסתגל לצרכי המשתמש ולספק תובנות ופעולות בעלות ערך.

בעוד ששימוש בכלים מציע אפשרויות עצומות, חשוב להיות מודעים לאתגרים ושיקולים פוטנציאליים. היבט מרכזי אחד הוא ניהול המורכבות של אינטראקציות בין כלים והבטחת היציבות והאמינות של המערכת הכוללת. עליך לטפל בתרחישים שבהם קריאות לכלים עלולות להיכשל, להחזיר תוצאות בלתי צפויות, או להשפיע על הביצועים. בנוסף, עליך לשקול אמצעי אבטחה ובקרת גישה כדי למנוע שימוש לא מורשה או זדוני בכלים. טיפול נאות בשגיאות, תיעוד ומנגנוני ניטור הם קריטיים לשמירה על השלמות והביצועים של האפליקציה מונעת-AI שלך.

בזמן שאתם חוקרים את האפשרויות של שימוש בכלים בפרויקטים שלכם, זכרו להתחיל עם מטרות ברורות, לעצב הגדרות כלים מובנות היטב, ולבצע איטרציות בהתבסס על משוב ותוצאות. עם הגישה והמחשבה הנכונה, שימוש בכלים יכול לפתוח רמות חדשות של חדשנות וערך ביישומים מבוססי בינה מלאכותית שלכם

עיבוד זרם

הזרמת נתונים על גבי HTTP, המכונה גם אירועים הנשלחים מהשרת (SSE), היא מנגנון שבו השרת שולח נתונים באופן רציף ללקוח ככל שהם זמינים, מבלי שהלקוח יצטרך לבקש אותם במפורש. מכיוון שתגובת הבינה המלאכותית נוצרת באופן הדרגתי, הגיוני לספק חוויית משתמש מגיבה על ידי הצגת הפלט של הבינה המלאכותית תוך כדי יצירתו. ולמעשה, כל ממשקי ה-API של ספקי הבינה המלאכותית שאני מכיר מציעים תגובות הזרמה כאפשרות בנקודות הקצה שלהם להשלמה.

הסיבה שפרק זה מופיע כאן בספר, מיד אחרי שימוש בכלים היא בגלל העוצמה שיכולה להיות בשילוב השימוש בכלים עם תגובות בינה מלאכותית חיות למשתמשים. שילוב זה מאפשר חוויות דינמיות ואינטראקטיביות שבהן הבינה המלאכותית יכולה לעבד קלט משתמש,

להשתמש בכלים ופונקציות שונות לפי שיקול דעתה, ואז לספק תגובות בזמן אמת.

כדי להשיג אינטראקציה חלקה זו, עליך לכתוב מטפלי זרם שיכולים לשלוח קריאות לפונקציות כלי המופעלות על ידי הבינה המלאכותית וכן פלט טקסט רגיל למשתמש הקצה. הצורך בלולאה לאחר עיבוד פונקציית כלי מוסיף אתגר מעניין למשימה.

מימוש ReplyStream

כדי להדגים כיצד ניתן ליישם עיבוד זרם, פרק זה יעמיק בגרסה מפושטת של מחלקת ReplyStream המשמשת ב-Olympia. ניתן להעביר מופעים של מחלקה זו כפרמטר stream בספריות לקוח בינה מלאכותית כמו ruby-openai ו-openrouter

הנה כיצד אני משתמש ב-ReplyStream ב-PromptSubscriber של Olympia, אשר מאזין באמצעות Wisper ליצירת הודעות משתמש חדשות.

```
1  class PromptSubscriber
2    include Raix::ChatCompletion
3    include Raix::PromptDeclarations
4
5    omitted... declarations other many #
6
7    prompt text: -> { user_message.content },
8           stream: -> { ReplyStream.new(self) },
9           until: -> { bot_message.complete? }
10
11   def message_created(message) Wisper by invoked #
12     return unless message.role.user? && message.content?
13
14     omitted... implementation the of rest #
```

בנוסף להפניית context למנוי ההנחיה שיצר אותה, המחלקה ReplyStream מכילה גם משתני מופע לאחסון מאגר זמני של מידע שהתקבל, ומערכים למעקב אחר שמות פונקציות וארגומנטים שהופעלו במהלך עיבוד הזרם.

```ruby
class ReplyStream
  attr_accessor :buffer, :f_name, :f_arguments, :context

  delegate :bot_message, :dispatch, to: :context

  def initialize(context)
    self.context = context
    self.buffer = []
    self.f_name = []
    self.f_arguments = []
  end

  def call(chunk, bytesize = nil)
    ... #
  end

  ... #
end
```

שיטת ה-initialize מגדירה את המצב ההתחלתי של מופע ה-ReplyStream, מאתחלת את החוצץ, ההקשר ומשתנים נוספים.

שיטת ה-call היא נקודת הכניסה העיקרית לעיבוד הנתונים הזורמים. היא מקבלת chunk של נתונים (המיוצג כטבלת גיבוב) ופרמטר אופציונלי bytesize, אשר בדוגמה שלנו אינו בשימוש. בתוך שיטה זו, המחלקה משתמשת בהתאמת תבניות כדי לטפל בתרחישים שונים בהתבסס על המבנה של המקטע שהתקבל.

קריאה ל-keys_symbolize_deep על המקטע עוזרת להפוך את התאמת התבניות לאלגנטית יותר, בכך שהיא מאפשרת לנו לעבוד עם סמלים במקום מחרוזות.

```ruby
def call(chunk, _bytesize)
  case chunk.deep_symbolize_keys

  in { name function match #
    choices: [
      {
        delta: {
          tool_calls: [
            { index: index, function: {name: name} }
          ]
        }
      }
    ] }

    f_name[index] = name
```

התבנית הראשונה שאנחנו מחפשים היא קריאת כלי יחד עם שם הפונקציה המשויך אליה. אם אנחנו מזהים אחת כזו, אנחנו מכניסים אותה לתוך המערך name_f. אנחנו מאחסנים שמות פונקציות במערך ממופתח, מכיוון שהמודל מסוגל לבצע קריאות פונקציה מקבילות, ולשלוח יותר מפונקציה אחת לביצוע בו-זמנית.

> קריאת פונקציות מקבילה היא היכולת של מודל בינה מלאכותית לבצע מספר קריאות פונקציה יחד, המאפשרת לתוצאות והשפעות של קריאות פונקציה אלה להתרחש במקביל. זה שימושי במיוחד אם הפונקציות לוקחות זמן רב, ומפחית את סבבי התקשורת עם ה-API, מה שבתורו יכול לחסוך כמות משמעותית של צריכת טוקנים.

לאחר מכן עלינו לחפש את הארגומנטים המתאימים לקריאות הפונקציה.

```
in { arguments match #
  choices: [
    {
      delta: {
        tool_calls: [
          {
            index: index, function: {arguments: argument }
          }
        ]
      }
    }
  ]}

  f_arguments[index] ||= "" already not if initialize #
  f_arguments[index] << argument
```

בדומה לאופן שבו טיפלנו בשמות הפונקציות, אנחנו מאחסנים את הארגומנטים במערך ממופתח.

בשלב הבא, אנחנו מחפשים הודעות רגילות המיועדות למשתמש, אשר יגיעו מהשרת אסימון אחר אסימון ויוקצו למשתנה content_new. עלינו גם לעקוב אחר reason_finish. הוא יהיה nil עד לחלק האחרון של רצף הפלט.

```
in {
  choices: [
    { delta: {content: new_content}, finish_reason: finish_reason }
  ]}

  here... user the to chunk every transmit could you #
  buffer << new_content.to_s

  if finish_reason.present?
    finalize
  elsif new_content.to_s.match?(/\n\n/)
    send_to_client paragraph per once transmit and buffer ...or #
  end
```

חשוב לציין שאנו מוסיפים ביטוי התאמת תבנית כדי לטפל בהודעות שגיאה הנשלחות על ידי ספק מודל הבינה המלאכותית. בסביבות פיתוח מקומיות, אנו זורקים חריגה, אך בסביבת הייצור, אנו מתעדים את השגיאה ומסיימים.

```
  in { error: { message: } }
    if Rails.env.local?
      raise message
    else
      Honeybadger.notify(" Error: AI#{message}")
      finalize
    end
```

סעיף ה-else האחרון של case יתבצע אם אף אחת מהתבניות הקודמות לא התאימה. זהו פשוט אמצעי בטיחות כדי שאם מודל הבינה המלאכותית יתחיל לשלוח לנו קטעים לא מזוהים, נדע על כך.

```
    else
      Honeybadger.notify(" Chunk: Unrecognized#{chunk}")
    end
  end
```

המתודה client_to_send אחראית לשליחת התוכן שבמאגר אל הלקוח. היא בודקת שהמאגר אינו ריק, מעדכנת את תוכן הודעת הבוט, מרנדרת את הודעת הבוט, ושומרת את התוכן במסד הנתונים כדי להבטיח שמירת המידע.

```
def send_to_client
  whitespace pure process to need no #
  return if buffer.join.squish.blank?

  message bot the on content buffer the set #
  content = buffer.join
  bot_message.content = content

  data lose never we that so database to save #
  correctly terminate doesn't stream the if even #
  bot_message.update_column(:content, content)

  websocket via content update #
  ConversationRenderer.update(bot_message)
end
```

שיטת ה-finalize נקראת כאשר עיבוד הזרם מסתיים. היא מבצעת את קריאות הפונקציה אם התקבלו כאלה במהלך הזרם, מעדכנת את הודעת הבוט עם התוכן הסופי ומידע רלוונטי אחר, ומאפסת את היסטוריית קריאות הפונקציה

```
def finalize
  if f_name.any?
    f_name.each_with_index do |name, index|
      implemented it's wherever function the calling of care takes #
      dispatch(name:, arguments: JSON.parse(f_arguments[index]))
    end

    history call function the reset #
    f_name.clear
    f_arguments.clear
  else
    content = buffer.join.presence
    bot_message.update!(content:, complete: true)
    ConversationRenderer.update(bot_message)
  end
end
```

אם המודל מחליט לקרוא לפונקציה, עליך "לשגר" את קריאת הפונקציה (שם וארגומנטים) באופן שיגרום לביצועה ולהוספת הודעות call_function ו-result_function לתמליל השיחה

מניסיוני, עדיף לטפל ביצירת הודעות הפונקציה במקום אחד בבסיס הקוד, במקום להסתמך על מימושי הכלים. זה לא רק יותר נקי, אלא יש לכך גם סיבה מעשית חשובה: אם מודל הבינה המלאכותית קורא לפונקציה, ולא רואה את הודעות הקריאה והתוצאה בתמליל כשאתה חוזר בלולאה, הוא יקרא לאותה פונקציה שוב. באופן פוטנציאלי לנצח. זכור שהבינה המלאכותית היא לחלוטין חסרת מצב , ולכן אם לא תשקף בחזרה את קריאות הפונקציה הללו, הן פשוט לא התרחשו מבחינתה.

```
PromptSubscriber#dispatch #

def dispatch(name:, arguments:)
 transcript conversation the to message function_call a adds #
 result returns and tool to dispatches plus #
 conversation.function_call!(name, arguments).then do |result|
  transcript the to message result function add #
  conversation.function_result!(name, result)
 end
end
```

ניקוי היסטוריית קריאות הפונקציות לאחר השליחה חשוב באותה מידה כמו להבטיח שהקריאה והתוצאות מגיעות לתמליל שלך, כדי שלא תמשיך לקרוא לאותן פונקציות שוב ושוב בכל פעם שאתה מבצע לולאה.

לולאת ה"שיחה"

אני ממשיך להזכיר לולאות, אבל אם אתה חדש בקריאה לפונקציות, ייתכן שלא ברור למה אנחנו צריכים לולאה. הסיבה היא שברגע שה-AI "מבקש" ממך להפעיל פונקציות כלי מטעמו, הוא יפסיק להגיב. באחריותך להפעיל את הפונקציות האלה, לאסוף את התוצאות, להוסיף את התוצאות לתמליל, ואז להגיש מחדש את ההנחיה המקורית כדי לקבל סדרה חדשה של קריאות פונקציה או תוצאות המיועדות למשתמש.

במחלקת PromptSubscriber, אנחנו משתמשים בשיטת prompt ממודול PromptDeclarations כדי להגדיר את התנהגות לולאת השיחה. הפרמטר until מוגדר ל--> { message.complete?_bot }, מה שאומר שהלולאה תמשיך עד ש-message_bot מסומן כהושלם.

```
prompt text: -> { user_message.content },
    stream: -> { ReplyStream.new(self) },
    until: -> { bot_message.complete? }
```

אבל מתי message_bot מסומן כהושלם? אם שכחת, חזור להסתכל על שורה 13 של המתודה finalize.

הבה נסקור את כל לוגיקת עיבוד הזרם.

1. ה-PromptSubscriber מקבל הודעת משתמש חדשה דרך המתודה created_message, אשר מופעלת על ידי מערכת הפרסום/הרשמה Wisper בכל פעם שהמשתמש הסופי יוצר הנחיה חדשה.
2. מתודת ה-prompt מגדירה באופן הצהרתי את התנהגות לוגיקת השלמת הצ'אט עבור ה-PromptSubscriber. מודל הבינה המלאכותית יבצע השלמת צ'אט עם תוכן הודעת המשתמש, מופע חדש של ReplyStream כפרמטר הזרם, ותנאי הלולאה המוגדר.
3. מודל הבינה המלאכותית מעבד את ההנחיה ומתחיל לייצר תגובה. כאשר התגובה מוזרמת, המתודה call של מופע ה-ReplyStream מופעלת עבור כל חלק מידע.
4. אם מודל הבינה המלאכותית מחליט לקרוא לפונקציית כלי, שם הפונקציה והארגומנטים נשלפים מהחלק ונשמרים במערכים name_f ו-arguments_f בהתאמה.
5. אם מודל הבינה המלאכותית מייצר תוכן המיועד למשתמש, הוא נשמר בחוצץ ונשלח ללקוח באמצעות המתודה client_to_send.
6. כאשר עיבוד הזרם מושלם, המתודה finalize נקראת. אם הופעלו פונקציות כלי במהלך הזרם, הן נשלחות באמצעות המתודה dispatch של ה-PromptSubscriber.
7. המתודה dispatch מוסיפה הודעת call_function לתמליל השיחה, מפעילה את פונקציית הכלי המתאימה, ומוסיפה הודעת result_function לתמליל עם תוצאת קריאת הפונקציה.
8. לאחר שליחת פונקציות הכלי, היסטוריית קריאות הפונקציה מנוקה כדי למנוע קריאות פונקציה כפולות בלולאות הבאות.
9. אם לא הופעלו פונקציות כלי, המתודה finalize מעדכנת את ה-message_bot עם התוכן הסופי, מסמנת אותו כהושלם, ושולחת את ההודעה המעודכנת ללקוח.

10. תנאי הלולאה -> { message.complete?_bot } מוערך. אם ה-message_bot לא מסומן כהושלם, הלולאה ממשיכה, וההנחיה המקורית נשלחת שוב עם תמליל השיחה המעודכן.
11. שלבים 3-10 חוזרים על עצמם עד ש-message_bot מסומן כהושלם, מה שמציין שמודל ה-AI סיים לייצר את התגובה שלו ואין צורך בהפעלת פונקציות כלים נוספות.

באמצעות מימוש לולאת השיחה הזו, אתה מאפשר למודל ה-AI לקיים אינטראקציה הדדית עם האפליקציה, להפעיל פונקציות כלים לפי הצורך ולייצר תגובות למשתמש עד שהשיחה מגיעה לסיום טבעי.

השילוב של עיבוד זרם ולולאת השיחה מאפשר חוויות דינמיות ואינטראקטיביות מבוססות AI, שבהן מודל ה-AI יכול לעבד קלט משתמש, להשתמש בכלים ופונקציות שונות, ולספק תגובות בזמן אמת בהתבסס על הקשר השיחה המתפתח.

המשכיות אוטומטית

חשוב להיות מודעים למגבלות הפלט של ה-AI. לרוב המודלים יש מספר מקסימלי של טוקנים שהם יכולים לייצר בתגובה בודדת, אשר נקבע על ידי הפרמטר tokens_max. אם מודל ה-AI מגיע למגבלה זו בזמן יצירת תגובה, הוא יעצור באופן פתאומי ויציין שהפלט נקטע.

בתגובת הזרם מה-API של פלטפורמת ה-AI, ניתן לזהות מצב זה על ידי בדיקת המשתנה reason_finish במקטע. אם ה-reason_finish מוגדר כ-"length" (או ערך מפתח אחר ספציפי למודל), זה אומר שהמודל הגיע למגבלת הטוקנים המקסימלית שלו במהלך היצירה והפלט נקטע.

דרך אחת לטפל בתרחיש זה בצורה חלקה ולספק חוויית משתמש חלקה, היא ליישם מנגנון המשכיות אוטומטית בלוגיקת עיבוד הזרם שלך. על ידי הוספת התאמת תבנית לסיבות סיום הקשורות לאורך, אתה יכול לבחור ללולאה ולהמשיך אוטומטית את הפלט מהנקודה שבה הוא נעצר.

הנה דוגמה מפושטת במכוון כיצד ניתן לשנות את המתודה call במחלקה ReplyStream כדי לתמוך בהמשכיות אוטומטית:

```ruby
LENGTH_STOPS = %w[MAX_TOKENS length]

def call(chunk, _bytesize)
  case chunk.deep_symbolize_keys
    ... #

    in {
      choices: [
        { delta: {content: new_content},
          finish_reason: finish_reason } ] }

      buffer << new_content.to_s

      if finish_reason.blank?
        send_to_client if new_content.to_s.match?(/\n\n/)
      elsif LENGTH_STOPS.include?(finish_reason)
        continue_cutoff
      else
        finalize
      end

    ... #
  end
end

private

def continue_cutoff
  conversation.bot_message!(buffer.join, visible: false)
  conversation.user_message!("continue please", visible: false)
  bot_message.update_column(:created_at, Time.current)
end
```

בגרסה המותאמת הזו, כאשר ה-reason_finish מציין פלט קטוע, במקום לסיים את הזרם, אנו מוסיפים זוג הודעות לתמליל מבלי לסיים אותו, מעבירים את ההודעה המקורית הנראית למשתמש ל"תחתית" התמליל על ידי עדכון מאפיין ה-at_created שלה, ואז מאפשרים ללולאה להתרחש, כך שה-AI ממשיך לייצר תוכן מהנקודה בה נעצר.

זכרו כי נקודת הקצה של השלמת ה-AI היא חסרת מצב. היא "יודעת" רק מה שאתם אומרים לה דרך התמליל. במקרה זה, הדרך שבה אנו מתקשרים ל-AI שהוא נקטע היא על

ידי הוספת הודעות "בלתי נראות" (למשתמש הקצה) לתמליל. זכרו, עם זאת, שזוהי דוגמה פשוטה במתכוון. יישום אמיתי יצטרך לבצע ניהול תמליל נוסף כדי להבטיח שלא נבזבז טוקנים ו/או נבלבל את ה-AI עם הודעות עוזר כפולות בתמליל.

יישום אמיתי של המשכיות אוטומטית צריך גם לכלול את מה שנקרא לוגיקת מפסק זרם כדי למנוע לולאות בלתי מבוקרות. הסיבה היא שבהינתן סוגים מסוימים של הנחיות משתמש והגדרות tokens_max נמוכות, ה-AI עלול להמשיך ללולאה פלט הנראה למשתמש ללא הפסקה.

A> בקחו בחשבון שכל לולאה דורשת בקשה נפרדת, ושכל בקשה צורכת את התמליל המלא שלכם שוב. אתם בהחלט צריכים לשקול את היתרונות והחסרונות בין חוויית המשתמש לבין השימוש ב-API כאשר אתם מחליטים אם ליישם המשכיות אוטומטית באפליקציה שלכם. המשכיות אוטומטית בפרט יכולה להיות מסוכנת מבחינת עלויות, במיוחד בשימוש במודלים מסחריים פרימיום.

סיכום

עיבוד זרם הוא היבט קריטי בבניית אפליקציות מבוססות AI המשלבות שימוש בכלים עם תגובות AI בזמן אמת. על ידי טיפול יעיל בנתוני הזרימה ממשקי API של פלטפורמות AI, אתם יכולים לספק חוויית משתמש חלקה ואינטראקטיבית, לטפל בתגובות גדולות, לייעל את השימוש במשאבים, ולטפל בשגיאות בצורה מסודרת.

מחלקת Conversation::ReplyStream המסופקת מדגימה כיצד ניתן ליישם עיבוד זרם באפליקציית Ruby באמצעות התאמת תבניות וארכיטקטורה מונעת אירועים. על ידי הבנה וניצול טכניקות עיבוד זרם, אתם יכולים לשחרר את הפוטנציאל המלא של שילוב AI באפליקציות שלכם ולספק חוויות משתמש עוצמתיות ומעורבות.

נתונים בעלי יכולת ריפוי עצמי

נתונים בעלי יכולת ריפוי עצמי היא גישה רבת עוצמה להבטחת שלמות הנתונים, עקביותם ואיכותם ביישומים באמצעות ניצול היכולות של מודלים שפתיים גדולים (LLMs). קטגוריה זו של תבניות מתמקדת ברעיון של שימוש בבינה מלאכותית לזיהוי אוטומטי, אבחון ותיקון של חריגות בנתונים, חוסר עקביות או שגיאות, ובכך מפחיתה את העומס על המפתחים ושומרת על רמה גבוהה של אמינות הנתונים.

בליבן, תבניות הנתונים בעלי יכולת הריפוי העצמי מכירות בכך שהנתונים הם נשמת אפה של כל אפליקציה, והבטחת הדיוק והשלמות שלהם היא קריטית לתפקוד תקין ולחוויית המשתמש של האפליקציה. עם זאת, ניהול ושמירה על איכות הנתונים יכולים להיות משימה מורכבת וצורכת זמן, במיוחד כאשר האפליקציות גדלות בהיקף ובמורכבות. כאן נכנס לתמונה כוחה

של הבינה המלאכותית.

בתבניות הנתונים בעלי יכולת הריפוי העצמי, עובדי בינה מלאכותית מועסקים כדי לנטר ולנתח באופן רציף את נתוני האפליקציה שלך. למודלים אלה יש את היכולת להבין ולפרש דפוסים, קשרים וחריגות בתוך הנתונים. באמצעות ניצול יכולות עיבוד והבנת השפה הטבעית שלהם, הם יכולים לזהות בעיות או חוסר עקביות פוטנציאליים בנתונים ולנקוט בפעולות מתאימות כדי לתקן אותם.

תהליך הריפוי העצמי של הנתונים כולל בדרך כלל מספר שלבים מרכזיים:

1. **ניטור נתונים:** עובדי הבינה המלאכותית מנטרים באופן קבוע את זרמי הנתונים, מסדי הנתונים או מערכות האחסון של האפליקציה, ומחפשים סימנים כלשהם לחריגות, חוסר עקביות או שגיאות. לחלופין, ניתן להפעיל רכיב בינה מלאכותית בתגובה לחריגה.
2. **זיהוי חריגות:** כאשר מתגלה בעיה, עובד הבינה המלאכותית מנתח את הנתונים בפירוט כדי לזהות את האופי וההיקף הספציפי של הבעיה. זה יכול לכלול זיהוי ערכים חסרים, פורמטים לא עקביים, או נתונים שמפרים כללים או אילוצים מוגדרים מראש.
3. **אבחון ותיקון:** לאחר זיהוי הבעיה, עובד הבינה המלאכותית משתמש בידע ובהבנה שלו בתחום הנתונים כדי לקבוע את דרך הפעולה המתאימה. זה יכול לכלול תיקון אוטומטי של הנתונים, מילוי ערכים חסרים, או סימון הבעיה להתערבות אנושית במידת הצורך.
4. **למידה מתמשכת (אופציונלי, תלוי במקרה השימוש):** כאשר עובד הבינה המלאכותית שלך נתקל ופותר בעיות נתונים שונות, הוא יכול לייצר פלט המתאר מה קרה וכיצד הגיב. מטא-נתונים אלה יכולים להיות מוזנים לתהליכי למידה המאפשרים לך (ואולי למודל הבסיסי, באמצעות כוונון עדין) להיות יעיל ואפקטיבי יותר לאורך זמן בזיהוי ופתרון חריגות בנתונים.

באמצעות זיהוי ותיקון אוטומטי של בעיות בנתונים, תוכלו להבטיח שהאפליקציה שלכם פועלת על בסיס נתונים איכותיים ואמינים. זה מפחית את הסיכון לשגיאות, חוסר עקביות, או באגים הקשורים לנתונים שמשפיעים על פונקציונליות האפליקציה או על חוויית המשתמש.

ברגע שיש לכם עובדי בינה מלאכותית שמטפלים במשימת ניטור ותיקון הנתונים, תוכלו למקד את מאמציכם בהיבטים קריטיים אחרים של האפליקציה. זה חוסך זמן ומשאבים שהיו מושקעים אחרת בניקוי ותחזוקת נתונים ידניים. למעשה, ככל שהאפליקציות שלכם

גדלות בהיקף ומורכבות, ניהול איכות הנתונים באופן ידני הופך למאתגר יותר ויותר. תבניות "נתונים המתרפאים מעצמם" מתרחבות ביעילות על ידי ניצול כוחה של הבינה המלאכותית לטיפול בכמויות גדולות של נתונים וזיהוי בעיות בזמן אמת.

בשל טבעם, מודלים של בינה מלאכותית יכולים להסתגל לשינויים בתבניות נתונים, סכמות, או דרישות לאורך זמן עם מעט או ללא פיקוח. כל עוד ההנחיות שלהם מספקות הכוונה מספקת, במיוחד בנוגע לתוצאות הרצויות, האפליקציה שלכם עשויה להיות מסוגלת להתפתח ולטפל בתרחישי נתונים חדשים מבלי לדרוש התערבות ידנית נרחבת או שינויי קוד.

תבניות הנתונים המתרפאים מעצמם מתיישרות היטב עם הקטגוריות האחרות של תבניות שדנו בהן, כמו "ריבוי עובדים". ניתן לראות ביכולת הריפוי העצמי של הנתונים כסוג מיוחד של עובד המתמקד ספציפית בהבטחת איכות ושלמות הנתונים. סוג זה של עובד פועל לצד עובדי בינה מלאכותית אחרים, כשכל אחד תורם להיבטים שונים של פונקציונליות האפליקציה.

יישום תבניות נתונים המתרפאים מעצמם בפועל דורש תכנון וסינטגרציה קפדניים של מודלי בינה מלאכותית לתוך ארכיטקטורת האפליקציה. בגלל הסיכונים של אובדן נתונים ושחיתות, עליכם להגדיר הנחיות ברורות לאופן השימוש בטכניקה זו. כדאי גם לשקול גורמים כמו ביצועים, יכולת הרחבה, ואבטחת נתונים.

מקרה בוחן מעשי: תיקון JSON שבור

אחת הדרכים המעשיות והנוחות ביותר לנצל נתונים המתרפאים מעצמם היא גם פשוטה מאוד להסבר: תיקון JSON שבור.

ניתן ליישם טכניקה זו לאתגר הנפוץ של התמודדות עם נתונים לא מושלמים או לא עקביים שנוצרו על ידי LLMs, כמו JSON שבור, והיא מספקת גישה לזיהוי ותיקון אוטומטי של בעיות אלה.

ב-Olympia אני נתקל באופן קבוע בתרחישים שבהם LLMs מייצרים נתוני JSON שאינם תקפים לחלוטין. זה יכול לקרות ממגוון סיבות, כמו הוספת הערות על ידי ה-LLM לפני או אחרי קוד ה-JSON עצמו, או הכנסת שגיאות תחביר כמו פסיקים חסרים או מרכאות כפולות שלא

טופלו כראוי. בעיות אלה יכולות להוביל לשגיאות ניתוח ולגרום להפרעות בפונקציונליות של האפליקציה.

כדי לטפל בבעיה זו, מימשתי פתרון מעשי בצורת מחלקת JsonFixer. מחלקה זו מגלמת את תבנית Self-Healing" "Data על ידי קבלת ה-JSON השבור כקלט ושימוש ב-LLM לתיקונו, תוך שמירה על מרב המידע והכוונה המקורית ככל האפשר.

```
class JsonFixer
  include Raix::ChatCompletion

  def call(bad_json, error_message)
    raise "provided data No" if bad_json.blank? || error_message.blank?

    transcript << {
        system: "parse a generated that JSON user-provided Consider
the preserving while it fix to best your Do exception.
possible. as much as intent and content original " }
    transcript << { user: bad_json }
    transcript << { assistant: "message? error the is What"}
    transcript << { user: error_message }
    transcript << { assistant: "JSON corrected the is Here\n‘json“\n" }

    self.stop = ["“‘"]

    chat_completion(json: true)
  end

  def model
    "mistralai/mixtral-8x7b-instruct:nitro"
  end
end
```

שים לב כיצד JsonFixer משתמש ב-Ventriloquist כדי להנחות את תגובות הבינה המלאכותית.

תהליך ריפוי עצמי של נתוני JSON עובד באופן הבא:

1. **ייצור JSON:** נעשה שימוש ב-LLM כדי לייצר נתוני JSON בהתבסס על הנחיות או דרישות מסוימות. עם זאת, בשל טבעם של מודלי LLM, ה-JSON המיוצר לא תמיד יהיה תקין לחלוטין. מנתח ה-JSON כמובן יעלה שגיאת ParserError אם תספק לו JSON לא תקין.

```
begin
  JSON.parse(llm_generated_json)
rescue JSON::ParserError => e
  JsonFixer.new.call(llm_generated_json, e.message)
end
```

שימו לב שהודעת השגיאה מועברת גם לקריאה ל-JSONFixer כך שאין צורך להניח לחלוטין מה לא בסדר עם הנתונים, במיוחד מכיוון שהמנתח התחבירי לעתים קרובות יגיד לך בדיוק מה לא בסדר.

2. **תיקון מבוסס LLM:** המחלקה JSONFixer שולחת את ה-JSON השבור בחזרה ל-LLM, יחד עם הנחיה או הוראה ספציפית לתקן את ה-JSON תוך שמירה על המידע והכוונה המקוריים ככל האפשר. ה-LLM, שאומן על כמויות עצומות של נתונים ומבין תחביר JSON, מנסה לתקן את השגיאות וליצור מחרוזת JSON תקפה. גידור תגובות משמש להגבלת הפלט של ה-LLM, ואנו בוחרים ב-8x7B Mixtral כמודל ה-AI, מכיוון שהוא טוב במיוחד למשימה מסוג זה.
3. **אימות ואינטגרציה:** מחרוזת ה-JSON המתוקנת שהוחזרה על ידי ה-LLM מנותחת על ידי המחלקה JSONFixer עצמה, מכיוון שהיא קראה ל-completion(json:_chat) (true. אם ה-JSON המתוקן עובר אימות, הוא משולב בחזרה לתוך זרימת העבודה של האפליקציה, מה שמאפשר לאפליקציה להמשיך לעבד את הנתונים ללא הפרעה. ה-JSON הפגום "התרפא".

למרות שכתבתי ושכתבתי את היישום של JSONFixer משלי מספר פעמים, אני מסופק אם הזמן הכולל שהושקע בכל הגרסאות האלה הוא יותר משעה או שעתיים.

שימו לב ששימור הכוונה הוא מרכיב מפתח בכל תבנית של נתונים המתרפאים מעצמם. תהליך התיקון המבוסס על LLM מכוון לשמר את המידע והכוונה המקוריים של ה-JSON

שנוצר ככל האפשר. זה מבטיח שה-JSON המתוקן שומר על המשמעות הסמנטית שלו וניתן להשתמש בו ביעילות בהקשר של האפליקציה.

היישום המעשי הזה של גישת "נתונים המתרפאים מעצמם" ב-Olympia מדגים בבירור כיצד ניתן לנצל AI, ובמיוחד LLMs, כדי לפתור אתגרי נתונים מהעולם האמיתי. זה מציג את העוצמה של שילוב טכניקות תכנות מסורתיות עם יכולות AI לבניית אפליקציות חזקות ויעילות.

חוק פוסטל ותבנית "נתונים המתרפאים מעצמם"

"נתונים המתרפאים מעצמם", כפי שמודגם על ידי המחלקה JSONFixer, מתיישר היטב עם העיקרון הידוע כחוק פוסטל, המכונה גם עקרון החוסן. חוק פוסטל קובע:

"היה שמרן במה שאתה עושה, היה ליברלי במה שאתה מקבל מאחרים."

עיקרון זה, שנוסח במקור על ידי ג'ון פוסטל, חלוץ האינטרנט המוקדם, מדגיש את החשיבות של בניית מערכות שסובלניות לקלטים מגוונים או אפילו שגויים במקצת, תוך שמירה על היצמדות קפדנית לפרוטוקולים מוגדרים בעת שליחת פלטים.

בהקשר של "נתונים בעלי יכולת ריפוי עצמי", המחלקה JSONFixer מגלמת את חוק פוסטל בכך שהיא ליברלית בקבלת נתוני JSON שבורים או לא מושלמים שנוצרו על ידי מודלים שפתיים גדולים. היא אינה דוחה או נכשלת מיד כשהיא נתקלת ב-JSON שאינו תואם באופן מדויק לפורמט המצופה. במקום זאת, היא נוקטת בגישה סובלנית ומנסה לתקן את ה-JSON באמצעות כוחם של מודלים שפתיים גדולים.

בכך שהיא ליברלית בקבלת JSON לא מושלם, מחלקת JSONFixer מדגימה חוסן וגמישות. היא מכירה בכך שנתונים בעולם האמיתי מגיעים לעתים קרובות בצורות שונות ולא תמיד תואמים למפרטים קפדניים. על ידי טיפול מוצלח ותיקון סטיות אלה, המחלקה מבטיחה שהאפליקציה יכולה להמשיך לתפקד בצורה חלקה, גם בנוכחות נתונים לא מושלמים.

מצד שני, מחלקת JSONFixer מקיימת גם את ההיבט השמרני של חוק פוסטל כשמדובר בפלט. לאחר תיקון ה-JSON באמצעות מודלים שפתיים גדולים, המחלקה מוודאת את נכונות ה-JSON המתוקן כדי להבטיח שהוא תואם באופן מדויק לפורמט המצופה. היא שומרת על השלמות והדיוק של הנתונים לפני העברתם לחלקים אחרים של האפליקציה.

גישה שמרנית זו מבטיחה שהפלט של מחלקת JSONFixer הוא אמין ועקבי, מקדמת יכולת פעולה הדדית ומונעת התפשטות של שגיאות.

עובדות מעניינות על ג'ון פוסטל:

- ג'ון פוסטל (1943-1998) היה מדען מחשבים אמריקאי שמילא תפקיד מכריע בפיתוח האינטרנט. הוא היה ידוע בכינוי "אלוהי האינטרנט" בזכות תרומותיו המשמעותיות לפרוטוקולים ולתקנים הבסיסיים.
- פוסטל היה העורך של סדרת מסמכי Comments for Request (RFC), שהיא סדרה של הערות טכניות וארגוניות על האינטרנט. הוא כתב או היה שותף לכתיבת יותר מ-200 RFC, כולל הפרוטוקולים הבסיסיים כמו IP TCP, ו-SMTP.
- בנוסף לתרומותיו הטכניות, פוסטל היה ידוע בגישתו הצנועה והשיתופית. הוא האמין בחשיבות של השגת הסכמה ועבודה משותפת לבניית רשת חסינה ובעלת יכולת פעולה הדדית.
- פוסטל שימש כמנהל החטיבה לרשתות מחשבים ב-Institute Sciences Information (ISI) של אוניברסיטת דרום קליפורניה (USC) משנת 1977 ועד מותו הפתאומי בשנת 1998.
- בהכרה בתרומותיו העצומות, פוסטל זכה לאחר מותו בפרס טיורינג היוקרתי בשנת 1998, המכונה לעתים קרובות "פרס נובל למדעי המחשב".

מחלקת JSONFixer מקדמת חוסן, גמישות ויכולת פעולה הדדית, שהיו ערכי הליבה שפוסטל דגל בהם לאורך הקריירה שלו. על ידי בניית מערכות שסובלניות כלפי אי-שלמויות תוך שמירה על הקפדה על פרוטוקולים, אנו יכולים ליצור אפליקציות שהן חסינות ומסתגלות יותר בפני אתגרי העולם האמיתי.

שיקולים והתוויות נגד

יישומיות של גישות נתונים בעלי יכולת תיקון עצמי תלויה לחלוטין בסוג הנתונים שהאפליקציה שלך מטפלת בהם. יש סיבה מדוע אולי לא תרצה פשוט לבצע מונקיפאץ' ל-JSON.parse כדי לתקן אוטומטית את כל שגיאות ניתוח ה-JSON באפליקציה שלך: לא כל השגיאות יכולות או

צריכות להיות מתוקנות אוטומטית.

תיקון עצמי הוא מורכב במיוחד כאשר הוא משולב עם דרישות רגולטוריות או דרישות תאימות הקשורות לטיפול ועיבוד נתונים. לחלק מהתעשיות, כמו בריאות ופיננסים, יש תקנות כה מחמירות בנוגע לשלמות נתונים ויכולת ביקורת, שביצוע כל סוג של תיקון נתונים ב"קופסה שחורה" ללא פיקוח או תיעוד נאות עלול להפר תקנות אלה. חיוני להבטיח שכל טכניקות תיקון הנתונים העצמי שאתה מפתח יתאימו למסגרות החוקיות והרגולטוריות הרלוונטיות.

יישום טכניקות נתונים בעלי יכולת תיקון עצמי, במיוחד אלו המערבות מודלים של בינה מלאכותית, עשוי גם להשפיע משמעותית על ביצועי האפליקציה וניצול המשאבים. עיבוד כמויות גדולות של נתונים דרך מודלים של בינה מלאכותית לזיהוי ותיקון שגיאות יכול להיות אינטנסיבי מבחינה חישובית. חשוב להעריך את האיזון בין היתרונות של נתונים בעלי יכולת תיקון עצמי לבין עלויות הביצועים והמשאבים הנלוות.

עם זאת, בואו נצלול לגורמים המעורבים בהחלטה מתי והיכן ליישם גישה עוצמתית זו.

קריטיות הנתונים

בעת שקילת יישום טכניקות נתונים בעלי יכולת תיקון עצמי, חיוני להעריך את קריטיות הנתונים המעובדים. רמת הקריטיות מתייחסת לחשיבות ולרגישות של הנתונים בהקשר של האפליקציה שלך ותחום העסקים שלה.

במקרים מסוימים, תיקון אוטומטי של שגיאות נתונים עשוי שלא להיות מתאים, במיוחד אם הנתונים רגישים מאוד או יש להם השלכות משפטיות. לדוגמה, שקול את התרחישים הבאים:

1. **עסקאות פיננסיות:** באפליקציות פיננסיות, כמו מערכות בנקאות או פלטפורמות מסחר, דיוק הנתונים הוא בעל חשיבות עליונה. אפילו שגיאות קטנות בנתונים פיננסיים יכולות להוביל להשלכות משמעותיות, כמו יתרות חשבון שגויות, העברת כספים שגויה, או החלטות מסחר שגויות. במקרים אלה, תיקונים אוטומטיים ללא אימות וביקורת יסודיים עלולים להציג סיכונים בלתי קבילים.
2. **רשומות רפואיות:** אפליקציות בתחום הבריאות מטפלות בנתוני מטופלים רגישים וחסויים ביותר. אי-דיוקים ברשומות רפואיות יכולים להוביל להשלכות חמורות על בטיחות המטופל והחלטות טיפול. שינוי אוטומטי של נתונים רפואיים ללא פיקוח

ואימות נאות על ידי אנשי מקצוע מוסמכים בתחום הבריאות עלול להפר דרישות רגולטוריות ולסכן את רווחת המטופל.

3. **מסמכים משפטיים:** אפליקציות המטפלות במסמכים משפטיים, כמו חוזים, הסכמים, או הגשות לבית משפט, דורשות דיוק ושלמות קפדניים. אפילו שגיאות קטנות בנתונים משפטיים יכולות להוביל להשלכות משפטיות משמעותיות. תיקונים אוטומטיים בתחום זה עשויים שלא להיות מתאימים, מכיוון שהנתונים לעתים קרובות דורשים סקירה ואימות ידני על ידי מומחים משפטיים כדי להבטיח את תקפותם ואכיפתם.

בתרחישי נתונים קריטיים אלה, הסיכונים הכרוכים בתיקונים אוטומטיים עולים לרוב על היתרונות הפוטנציאליים. ההשלכות של הכנסת שגיאות או שינוי נתונים באופן שגוי עלולות להיות חמורות, ולהוביל להפסדים כספיים, חבות משפטית, ואפילו נזק לאנשים.

בעת טיפול בנתונים קריטיים במיוחד, חיוני לתת עדיפות לתהליכי אימות ותיקוף ידניים. פיקוח ומומחיות אנושיים הם קריטיים להבטחת דיוק ושלמות הנתונים. ניתן עדיין להשתמש בטכניקות ריפוי עצמי אוטומטיות כדי לסמן שגיאות או חוסר עקביות פוטנציאליים, אך ההחלטה הסופית לגבי תיקונים צריכה לכלול שיקול דעת ואישור אנושי.

עם זאת, חשוב לציין שלא כל הנתונים באפליקציה עשויים להיות באותה רמת קריטיות. באותה אפליקציה, ייתכנו תתי-קבוצות של נתונים שהם פחות רגישים או בעלי השפעה נמוכה יותר במקרה של שגיאות. במקרים כאלה, ניתן להחיל טכניקות ריפוי עצמי של נתונים באופן סלקטיבי על תתי-קבוצות ספציפיות אלה של נתונים, בעוד שנתונים קריטיים נשארים כפופים לאימות ידני.

המפתח הוא להעריך בקפידה את רמת הקריטיות של כל קטגוריית נתונים באפליקציה שלך ולהגדיר הנחיות ותהליכים ברורים לטיפול בתיקונים בהתבסס על הסיכונים וההשלכות הנלווים. על ידי הבחנה בין נתונים קריטיים (כגון ספרי חשבונות, רשומות רפואיות) לבין נתונים שאינם קריטיים (כגון כתובות דואר, אזהרות משאבים), ניתן למצוא איזון בין ניצול היתרונות של טכניקות ריפוי עצמי של נתונים במקום המתאים לבין שמירה על בקרה ופיקוח קפדניים במקום הנדרש.

בסופו של דבר, ההחלטה ליישם טכניקות ריפוי עצמי של נתונים על נתונים קריטיים צריכה להתקבל בהתייעצות עם מומחי תחום, יועצים משפטיים ובעלי עניין רלוונטיים אחרים. חיוני

לשקול את הדרישות הספציפיות, התקנות והסיכונים הקשורים לנתוני האפליקציה שלך ולהתאים את אסטרטגיות תיקון הנתונים בהתאם.

חומרת השגיאה

בעת יישום טכניקות ריפוי עצמי של נתונים, חשוב להעריך את חומרת ההשפעה של שגיאות הנתונים. לא כל השגיאות נוצרו שוות, והפעולה המתאימה עשויה להשתנות בהתאם לחומרת הבעיה.

חוסר עקביות קל או בעיות פורמט עשויים להתאים לתיקון אוטומטי. לדוגמה, עובד ריפוי נתונים עצמי שתפקידו לתקן JSON שבור יכול לטפל בפסיקים חסרים או מרכאות כפולות לא מוגנות מבלי לשנות משמעותית את המשמעות או המבנה של הנתונים. סוגים אלה של שגיאות הם לרוב פשוטים לתיקון ובעלי השפעה מינימלית על שלמות הנתונים הכוללת.

עם זאת, שגיאות חמורות יותר המשנות באופן יסודי את המשמעות או השלמות של הנתונים עשויות לדרוש גישה שונה. במקרים כאלה, תיקונים אוטומטיים עשויים לא להיות מספיקים, והתערבות אנושית עשויה להיות נחוצה כדי להבטיח את דיוק ותקפות הנתונים.

כאן נכנס לתמונה הרעיון של שימוש בבינה מלאכותית עצמה כדי לסייע בקביעת חומרת השגיאות. באמצעות ניצול היכולות של מודלים של בינה מלאכותית, אנחנו יכולים לתכנן עובדי נתונים בעלי יכולת תיקון עצמי שלא רק מתקנים שגיאות, אלא גם מעריכים את חומרת השגיאות הללו ומקבלים החלטות מושכלות כיצד לטפל בהן.

לדוגמה, הבה נתבונן בעובד נתונים בעל יכולת תיקון עצמי האחראי על תיקון חוסר עקביות בנתונים הזורמים למסד נתוני לקוחות. ניתן לתכנן את העובד כך שינתח את הנתונים ויזהה שגיאות פוטנציאליות, כגון מידע חסר או מידע סותר. עם זאת, במקום לתקן אוטומטית את כל השגיאות, ניתן לצייד את העובד בקריאות כלים נוספות המאפשרות לו לסמן שגיאות חמורות לבדיקה אנושית.

להלן דוגמה כיצד ניתן ליישם זאת:

```ruby
class CustomerDataReviewer
  include Raix::ChatCompletion
  include Raix::FunctionDeclarations

  attr_accessor :customer

  function :flag_for_review, reason: { type: "string" } do |params|
    AdminNotifier.review_request(customer, params[:reason])
  end

  def initialize(customer)
    self.customer = customer
  end

  def call(customer_data)
    transcript << {
      system: "identify to is task Your reviewer. data customer a are You
data. customer in inconsistencies correct and

> here... instructions additional <

the use ,review human require that errors severe encounter you If
intervention. manual for data the flag to tool ‘‘flag_for_review       ” }

    transcript << { user: customer.to_json }
    transcript << { assistant: "data: Reviewed/corrected\n‘json‘‘\n" }

    self.stop = ["‘‘‘"]

    chat_completion(json: true).then do |result|
      return if result.blank?

      customer.update(result)
    end
  end
end
```

בדוגמה זו, עובד ה-CustomerDataHealer מתוכנן לזהות ולתקן חוסר עקביות בנתוני לקוחות. שוב, אנו משתמשים בגידור תגובות ובדובב כדי לקבל פלט מובנה. חשוב לציין כי הנחיות המערכת של העובד כוללות הוראות להשתמש בפונקציית review_for_flag אם מתגלות

שגיאות חמורות.

כאשר העובד מעבד את נתוני הלקוח, הוא מנתח את הנתונים ומנסה לתקן חוסר עקביות כלשהו. אם העובד קובע כי השגיאות חמורות ודורשות התערבות אנושית, הוא יכול להשתמש בכלי review_for_flag כדי לסמן את הנתונים ולספק סיבה לסימון.

שיטת completion_chat נקראת עם :json true כדי לנתח את נתוני הלקוח המתוקנים כ-JSON. אין אפשרות ללולאה לאחר קריאת פונקציה, ולכן התוצאה תהיה ריקה אם review_for_flag הופעל. אחרת, הלקוח מתעדכן עם הנתונים שנבדקו ותוקנו במידת הצורך.

על ידי שילוב הערכת חומרת שגיאות והאפשרות לסמן נתונים לבדיקה אנושית, עובד הנתונים בעל הריפוי העצמי הופך לחכם ומסתגל יותר. הוא יכול לטפל בשגיאות קלות באופן אוטומטי תוך העברת שגיאות חמורות למומחים אנושיים להתערבות ידנית.

הקריטריונים הספציפיים לקביעת חומרת השגיאה ניתנים להגדרה בהנחיות העובד בהתבסס על הידע בתחום ודרישות העסק. גורמים כמו ההשפעה על שלמות הנתונים, הפוטנציאל לאובדן או שחיתות נתונים, וההשלכות של נתונים שגויים יכולים להילקח בחשבון בעת הערכת החומרה.

באמצעות שימוש בבינה מלאכותית להערכת חומרת שגיאות ומתן אפשרויות להתערבות אנושית, טכניקות נתונים בעלי ריפוי עצמי יכולות ליצור איזון בין אוטומציה לבין שמירה על דיוק הנתונים. גישה זו מבטיחה כי שגיאות קלות מתוקנות ביעילות בעוד ששגיאות חמורות מקבלות את תשומת הלב והמומחיות הנדרשת מבודקים אנושיים.

מורכבות התחום

בעת שקילת היישום של טכניקות נתונים בעלי ריפוי עצמי, חשוב להעריך את מורכבות תחום הנתונים והכללים המנחים את המבנה והקשרים שלו. מורכבות התחום יכולה להשפיע משמעותית על היעילות והישימות של גישות תיקון נתונים אוטומטיות.

טכניקות נתונים בעלי ריפוי עצמי עובדות היטב כאשר הנתונים עוקבים אחר תבניות ואילוצים מוגדרים היטב. בתחומים שבהם מבנה הנתונים פשוט יחסית והקשרים בין רכיבי הנתונים הם ישירים, ניתן ליישם תיקונים אוטומטיים ברמת ביטחון גבוהה. לדוגמה, תיקון בעיות

עיצוב או אכיפת אילוצי סוג נתונים בסיסיים יכולים לרוב להיות מטופלים ביעילות על ידי עובדי נתונים בעלי ריפוי עצמי.

עם זאת, ככל שמורכבות תחום הנתונים גדלה, גם האתגרים הקשורים לתיקון נתונים אוטומטי גדלים. בתחומים בעלי לוגיקה עסקית מורכבת, יחסים מורכבים בין ישויות נתונים, או כללים וחריגים ייחודיים לתחום, טכניקות ריפוי נתונים עצמי עלולות שלא לתפוס את כל הניואנסים ועלולות להוביל לתוצאות בלתי רצויות.

הבה נבחן דוגמה של תחום מורכב: מערכת מסחר פיננסית. בתחום זה, הנתונים כוללים מכשירים פיננסיים שונים, נתוני שוק, כללי מסחר ודרישות רגולטוריות. היחסים בין אלמנטי נתונים שונים יכולים להיות מורכבים, והכללים המסדירים את תקפות הנתונים ועקביותם יכולים להיות ספציפיים מאוד לתחום.

בתחום מורכב כזה, עובד ריפוי נתונים עצמי שתפקידו לתקן חוסר עקביות בנתוני מסחר יצטרך להבין לעומק את הכללים והאילוצים הייחודיים לתחום. הוא יצטרך להתחשב בגורמים כמו תקנות שוק, מגבלות מסחר, חישובי סיכונים והליכי סליקה. תיקונים אוטומטיים בהקשר זה עלולים שלא לתפוס את מלוא המורכבות של התחום ועלולים בשוגג להכניס שגיאות או להפר כללים ייחודיים לתחום.

כדי להתמודד עם אתגרי מורכבות התחום, ניתן לשפר טכניקות ריפוי נתונים עצמי על ידי שילוב ידע וכללים ייחודיים לתחום במודלים ובעובדים של הבינה המלאכותית. ניתן להשיג זאת באמצעות טכניקות כגון:

1. **אימון ייעודי לתחום:** ניתן לכוון או אפילו לבצע כוונון עדין של מודלי הבינה המלאכותית המשמשים לריפוי נתונים עצמי על מערכי נתונים ייחודיים לתחום שלוכדים את המורכבויות והכללים של התחום המסוים. על ידי חשיפת המודלים לנתונים ותרחישים מייצגים, הם יכולים ללמוד את הדפוסים, האילוצים והחריגים הספציפיים לתחום.
2. **אילוצים מבוססי כללים:** ניתן להעשיר עובדי ריפוי נתונים עצמי באילוצים מפורשים מבוססי כללים המקודדים ידע ייחודי לתחום. כללים אלה יכולים להיות מוגדרים על ידי מומחי תחום ומשולבים בתהליך תיקון הנתונים. מודלי הבינה המלאכותית יכולים אז להשתמש בכללים אלה כדי להנחות את החלטותיהם ולהבטיח עמידה בדרישות הייחודיות לתחום.

3. **שיתוף פעולה עם מומחי תחום:** בתחומים מורכבים, חיוני לערב מומחי תחום בתכנון ופיתוח של טכניקות ריפוי נתונים עצמי. מומחי תחום יכולים לספק תובנות חשובות לגבי מורכבויות הנתונים, כללי העסקים ומקרי הקצה האפשריים. ניתן לשלב את הידע שלהם במודלים ובעובדי הבינה המלאכותית כדי לשפר את הדיוק והאמינות של תיקוני נתונים אוטומטיים באמצעות דפוסי שילוב אדם בתהליך.
4. **גישה הדרגתית ואיטרטיבית:** כשמתמודדים עם תחומים מורכבים, לעתים קרובות כדאי לאמץ גישה הדרגתית ואיטרטיבית לריפוי נתונים עצמי. במקום לנסות לאוטומט תיקונים עבור כל התחום בבת אחת, להתמקד בתת-תחומים או קטגוריות נתונים ספציפיות שבהם הכללים והאילוצים מובנים היטב. להרחיב בהדרגה את היקף טכניקות הריפוי העצמי ככל שההבנה של התחום גדלה והטכניקות מוכיחות את עצמן כיעילות.

על ידי התחשבות במורכבות תחום הנתונים והטמעת ידע ספציפי לתחום בטכניקות נתונים בעלי יכולת תיקון עצמי, ניתן להשיג איזון בין אוטומציה לדיוק. חשוב להכיר בכך שנתונים בעלי יכולת תיקון עצמי אינם פתרון אחיד לכל המקרים, ושהגישה צריכה להיות מותאמת לדרישות ולאתגרים הספציפיים של כל תחום.

בתחומים מורכבים, גישה היברידית המשלבת טכניקות של נתונים בעלי יכולת תיקון עצמי יחד עם מומחיות ופיקוח אנושי עשויה להיות היעילה ביותר. תיקונים אוטומטיים יכולים לטפל במקרים שגרתיים ומוגדרים היטב, בעוד תרחישים מורכבים או חריגים יכולים להיות מסומנים לבדיקה והתערבות אנושית. גישה שיתופית זו מבטיחה שיתרונות האוטומציה מתממשים תוך שמירה על בקרה ודיוק הכרחיים בתחומי נתונים מורכבים.

יכולת הסבר ושקיפות

יכולת הסבר מתייחסת ליכולת להבין ולפרש את ההיגיון שעומד מאחורי ההחלטות שמתקבלות על ידי מודלים של בינה מלאכותית, בעוד ששקיפות כוללת מתן ראות ברורה לתהליך תיקון הנתונים.

בהקשרים רבים, שינויים בנתונים צריכים להיות ניתנים לביקורת ולהצדקה. בעלי עניין, כולל משתמשים עסקיים, מבקרים וגופים רגולטוריים, עשויים לדרוש הסברים מדוע נעשו תיקונים מסוימים בנתונים וכיצד מודלים של בינה מלאכותית הגיעו להחלטות אלה. זה חיוני במיוחד

בתחומים בהם דיוק ושלמות הנתונים הם בעלי השלכות משמעותיות, כמו פיננסים, בריאות ונושאים משפטיים.

כדי לתת מענה לצורך ביכולת הסבר ושקיפות, טכניקות של נתונים בעלי יכולת תיקון עצמי צריכות לכלול מנגנונים המספקים תובנות לגבי תהליך קבלת ההחלטות של מודלים של בינה מלאכותית. ניתן להשיג זאת באמצעות מספר גישות:

1. **שרשרת חשיבה:** בקשה מהמודל להסביר את חשיבתו "בקול רם" לפני ביצוע שינויים בנתונים עשויה לאפשר הבנה קלה יותר של תהליך קבלת ההחלטות ויכולה לייצר הסברים קריאים לאדם עבור התיקונים שבוצעו. המחיר הוא מעט יותר מורכבות בהפרדת ההסבר מפלט הנתונים המובנה, אותו ניתן לפתור על ידי...
2. **יצירת הסברים:** עובדי תיקון נתונים אוטומטיים יכולים להיות מצוידים ביכולת לייצר הסברים קריאים לאדם עבור התיקונים שהם מבצעים. ניתן להשיג זאת על ידי בקשה מהמודל להציג את תהליך קבלת ההחלטות שלו כהסברים קלים להבנה המשולבים בנתוניס עצמס. לדוגמה, עובד תיקון נתונים אוטומטי יכול לייצר דוח המדגיש את חוסר העקביות הספציפי בנתונים שזוהה, התיקונים שהוחלו, וההיגיון מאחורי תיקונים אלה.
3. **חשיבות מאפיינים:** ניתן להנחות מודלים של בינה מלאכותית עם מידע על חשיבות המאפיינים או התכונות השונות בתהליך תיקון הנתונים כחלק מההנחיות שלהם. הנחיות אלה, בתורן, יכולות להיות חשופות לבעלי העניין האנושיים. על ידי זיהוי הגורמים המרכזיים המשפיעים על החלטות המודל, בעלי העניין יכולים לקבל תובנות לגבי ההיגיון מאחורי התיקונים ולהעריך את תקפותם.
4. **תיעוד וביקורת:** יישום מנגנוני תיעוד וביקורת מקיפים הוא קריטי לשמירה על שקיפות בתהליך הנתונים בעלי יכולת התיקון העצמי. יש לתעד כל תיקון נתונים שמבוצע על ידי מודלים של בינה מלאכותית, כולל הנתונים המקוריים, הנתונים המתוקנים, והפעולות הספציפיות שננקטו. מעקב ביקורת זה מאפשר ניתוח רטרוספקטיבי ומספק תיעוד ברור של השינויים שנעשו בנתונים.
5. **גישת שילוב אנושי:** שילוב גישה המערבת מעורבות אנושית יכול לשפר את יכולת ההסבר והשקיפות של טכניקות נתונים בעלי יכולת תיקון עצמי. על ידי שיתוף מומחים אנושיים בסקירה ואימות של תיקונים שנוצרו על ידי בינה מלאכותית, ארגונים יכולים

להבטיח שהתיקונים מתואמים עם הידע התחומי ודרישות העסק. פיקוח אנושי מוסיף שכבת אחריותיות נוספת ומאפשר זיהוי של הטיות או שגיאות פוטנציאליות במודלים של הבינה המלאכותית.

6. **ניטור והערכה מתמשכים:** ניטור והערכה סדירים של ביצועי טכניקות הנתונים בעלי יכולת התיקון העצמי הם חיוניים לשמירה על שקיפות ואמון. באמצעות הערכת הדיוק והיעילות של מודלי הבינה המלאכותית לאורך זמן, ארגונים יכולים לזהות סטיות או חריגות ולנקוט בפעולות מתקנות. ניטור מתמשך עוזר להבטיח שתהליך הנתונים בעלי יכולת התיקון העצמי נשאר אמין ומתואם עם התוצאות הרצויות.

יכולת הסבר ושקיפות הן שיקולים קריטיים בעת יישום טכניקות נתונים בעלי יכולת תיקון עצמי. על ידי מתן הסברים ברורים לתיקוני נתונים, שמירה על מעקבי ביקורת מקיפים, ושילוב פיקוח אנושי, ארגונים יכולים לבנות אמון בתהליך הנתונים בעלי יכולת התיקון העצמי ולהבטיח שהשינויים שנעשים בנתונים הם מוצדקים ומתואמים עם יעדי העסק.

חשוב למצוא איזון בין יתרונות האוטומציה לבין הצורך בשקיפות. בעוד שטכניקות נתונים בעלי יכולת תיקון עצמי יכולות לשפר משמעותית את איכות הנתונים והיעילות, הן לא צריכות לבוא על חשבון אובדן הנראות והשליטה בתהליך תיקון הנתונים. על ידי תכנון עובדי נתונים בעלי יכולת תיקון עצמי עם התחשבות ביכולת הסבר ושקיפות, ארגונים יכולים לרתום את כוח הבינה המלאכותית תוך שמירה על רמת האחריותיות והאמון הנדרשת בנתונים.

השלכות בלתי מכוונות

בעוד שטכניקות נתונים בעלי יכולת תיקון עצמי מכוונות לשפר את איכות הנתונים ועקביותם, חשוב להיות מודעים לפוטנציאל של השלכות בלתי מכוונות. תיקונים אוטומטיים, אם לא מתוכננים ומנוטרים בקפידה, עלולים לשנות בטעות את המשמעות או ההקשר של הנתונים, מה שמוביל לבעיות במורד הזרם.

אחד הסיכונים העיקריים של נתונים בעלי יכולת תיקון עצמי הוא הכנסת הטיה או שגיאות בתהליך תיקון הנתונים. מודלים של בינה מלאכותית, כמו כל מערכת תוכנה אחרת, יכולים להיות כפופים להטיות הקיימות בנתוני האימון או כאלה שהוכנסו דרך תכנון האלגוריתמים. אם הטיות אלה לא מזוהות וממותנות, הן יכולות להתפשט דרך תהליך הנתונים בעלי יכולת התיקון העצמי ולהוביל לשינויי נתונים מוטים או שגויים.

לדוגמה, בואו נתבונן בעובד נתונים בעל יכולת ריפוי עצמי שתפקידו לתקן חוסר עקביות בנתונים דמוגרפיים של לקוחות. אם מודל הבינה המלאכותית למד הטיות מנתונים היסטוריים, כמו קישור מקצועות מסוימים או רמות הכנסה למגדרים או מוצאים אתניים ספציפיים, הוא עלול להגיע להנחות שגויות ולשנות את הנתונים באופן שמחזק הטיות אלה. הדבר עלול להוביל לפרופילי לקוחות לא מדויקים, החלטות עסקיות שגויות ותוצאות שעלולות להיות מפלות.

השלכה בלתי מכוונת אפשרית נוספת היא אובדן מידע או הקשר חשוב במהלך תהליך תיקון הנתונים. טכניקות של ריפוי עצמי של נתונים מתמקדות לרוב בתיקנון ונרמול נתונים כדי להבטיח עקביות. עם זאת, במקרים מסוימים, הנתונים המקוריים עשויים להכיל ניואנסים, יוצאים מן הכלל, או מידע הקשרי החשוב להבנת התמונה המלאה. תיקונים אוטומטיים המאכפים תיקנון באופן עיוור עלולים בשוגג להסיר או להסתיר מידע חשוב זה.

לדוגמה, דמיינו עובד נתונים בעל יכולת ריפוי עצמי האחראי לתיקון חוסר עקביות ברשומות רפואיות. אם העובד נתקל בהיסטוריה רפואית של מטופל עם מצב נדיר או תוכנית טיפול חריגה, הוא עשוי לנסות לנרמל את הנתונים כך שיתאימו לדפוס נפוץ יותר. עם זאת, בעשותו כן, הוא עלול לאבד את הפרטים הספציפיים וההקשר שהם קריטיים לייצוג מדויק של מצבו הייחודי של המטופל. אובדן מידע זה עלול להיות בעל השלכות חמורות על הטיפול במטופל וקבלת החלטות רפואיות.

כדי למזער את הסיכונים של השלכות בלתי מכוונות, חיוני לנקוט בגישה פרואקטיבית בעת תכנון ויישום טכניקות ריפוי עצמי של נתונים:

1. **בדיקה ותיקוף מעמיקים:** לפני הטמעת עובדי נתונים בעלי יכולת ריפוי עצמי בסביבת הייצור, חיוני לבדוק ולתקף ביסודיות את התנהגותם מול מגוון תרחישים. זה כולל בדיקה עם מערכי נתונים מייצגים המכסים מקרי קצה שונים, יוצאים מן הכלל והטיות פוטנציאליות. בדיקה קפדנית עוזרת לזהות ולטפל בהשלכות בלתי מכוונות לפני שהן משפיעות על נתונים בעולם האמיתי.
2. **ניטור והערכה מתמשכים:** יישום מנגנוני ניטור והערכה מתמשכים הוא חיוני לזיהוי ומיתון השלכות בלתי מכוונות לאורך זמן. סקירה קבועה של תוצאות תהליכי ריפוי עצמי של נתונים, ניתוח ההשפעה על מערכות במורד הזרם וקבלת החלטות, ואיסוף משוב מבעלי העניין יכולים לעזור בזיהוי השפעות שליליות ולהוביל לפעולות מתקנות

מהירות. אם לארגון שלכם יש לוחות מחוונים תפעוליים, כדאי כנראה להוסיף מדדים גלויים לעין הקשורים לשינויי נתונים אוטומטיים. הוספת התראות המחוברות לסטיות גדולות מפעילות שינוי נתונים רגילה היא כנראה רעיון עוד יותר טוב!

3. **פיקוח והתערבות אנושית:** שמירה על פיקוח אנושי ויכולת להתערב בתהליך הריפוי העצמי של הנתונים היא קריטית. בעוד שאוטומציה יכולה לשפר מאוד את היעילות, חשוב שמומחים אנושיים יסקרו ויתקפו את התיקונים שנעשו על ידי מודלים של בינה מלאכותית, במיוחד בתחומים קריטיים או רגישים. שיקול דעת אנושי ומומחיות בתחום יכולים לעזור בזיהוי וטיפול בהשלכות בלתי מכוונות שעלולות להתעורר.

4. **בינה מלאכותית מוסברת (XAI) ושקיפות:** כפי שנדון בתת-הסעיף הקודם, שילוב טכניקות של בינה מלאכותית מוסברת והבטחת שקיפות בתהליך הנתונים בעלי יכולת תיקון עצמי יכול לסייע במניעת תוצאות בלתי רצויות. על ידי מתן הסברים ברורים לתיקוני נתונים ושמירה על מעקב ביקורת מקיף, ארגונים יכולים להבין ולעקוב טוב יותר אחר ההיגיון שמאחורי השינויים שנעשו על ידי מודלים של בינה מלאכותית.
5. **גישה הדרגתית ואיטרטיבית:** אימוץ גישה הדרגתית ואיטרטיבית לנתונים בעלי יכולת תיקון עצמי יכול לסייע במזעור הסיכון לתוצאות בלתי רצויות. במקום להחיל תיקונים אוטומטיים על כל מערך הנתונים בבת אחת, מומלץ להתחיל עם תת-קבוצה של נתונים ולהרחיב בהדרגה את ההיקף ככל שהטכניקות מוכיחות את עצמן כיעילות ואמינות. גישה זו מאפשרת ניטור וכוונון זהירים לאורך הדרך, ומפחיתה את ההשפעה של תוצאות בלתי רצויות.
6. **שיתוף פעולה ומשוב:** מעורבות בעלי עניין מתחומים שונים ועידוד שיתוף פעולה ומשוב לאורך תהליך הנתונים בעלי יכולת תיקון עצמי יכולים לסייע בזיהוי וטיפול בתוצאות בלתי רצויות. קבלת משוב קבוע ממומחי תחום, צרכני מידע ומשתמשי קצה יכולה לספק תובנות חשובות לגבי ההשפעה המעשית של תיקוני הנתונים ולהדגיש בעיות שאולי נעלמו מהעין.

על ידי התמודדות פרואקטיבית עם הסיכון לתוצאות בלתי רצויות ויישום אמצעי הגנה מתאימים, ארגונים יכולים לנצל את היתרונות של טכניקות נתונים בעלי יכולת תיקון עצמי תוך מזעור השפעות שליליות אפשריות. חשוב להתייחס לנתונים בעלי יכולת תיקון עצמי

כתהליך איטרטיבי ושיתופי, תוך ניטור, הערכה ושיפור מתמשכים של הטכניקות כדי להבטיח שהן מתואמות עם התוצאות הרצויות ושומרות על שלמות ואמינות הנתונים.

בעת שקילת השימוש בדפוסי נתונים בעלי יכולת תיקון עצמי, חיוני להעריך בקפידה גורמים אלה ולשקול את היתרונות מול הסיכונים והמגבלות האפשריים. במקרים מסוימים, גישה היברידית המשלבת תיקונים אוטומטיים עם פיקוח והתערבות אנושית עשויה להיות הפתרון המתאים ביותר.

חשוב לציין גם שטכניקות נתונים בעלי יכולת תיקון עצמי לא צריכות להיחשב כתחליף למנגנוני אימות נתונים חזקים, טיהור קלט וטיפול בשגיאות. נהלים בסיסיים אלה נשארים קריטיים להבטחת שלמות ואבטחת הנתונים. נתונים בעלי יכולת תיקון עצמי צריכים להיחשב כגישה משלימה שיכולה להעצים ולשפר את האמצעים הקיימים.

בסופו של דבר, ההחלטה להשתמש בדפוסי נתונים בעלי יכולת תיקון עצמי תלויה בדרישות הספציפיות, באילוצים ובסדרי העדיפויות של היישום שלך. על ידי שקילה זהירה של השיקולים שתוארו לעיל והתאמתם למטרות ולארכיטקטורה של היישום שלך, תוכל לקבל החלטות מושכלות לגבי מתי וכיצד לנצל ביעילות טכניקות של נתונים בעלי יכולת תיקון עצמי.

יצירת תוכן הקשרי

תבניות יצירת תוכן הקשרי מנצלות את כוחם של מודלי שפה גדולים (LLMs) כדי ליצור תוכן דינמי ותלוי-הקשר בתוך יישומים. קטגוריה זו של תבניות מכירה בחשיבות של אספקת תוכן מותאם אישית ורלוונטי למשתמשים בהתבסס על הצרכים הספציפיים שלהם, העדפותיהם, ואפילו האינטראקציות הקודמות והנוכחיות שלהם עם היישום.

בהקשר של גישה זו, "תוכן" מתייחס הן לתוכן ראשי (כלומר, פוסטים בבלוג, מאמרים וכו') והן לתוכן-על, כגון המלצות לתוכן ראשי.

תבניות יצירת תוכן הקשרי יכולות למלא תפקיד מכריע בשיפור רמות המעורבות של המשתמשים שלך, מתן חוויות מותאמות אישית, ואוטומציה של משימות יצירת תוכן הן עבורך והן עבור המשתמשים שלך. באמצעות השימוש בתבניות שאנו מתארים בפרק זה, תוכל ליצור יישומים המייצרים תוכן באופן דינמי, תוך התאמה להקשר ולקלט בזמן אמת.

התבניות פועלות על ידי שילוב LLMs בפלטי היישום, החל מממשק המשתמש (המכונה לעתים "chrome"), ועד לדואר אלקטרוני וצורות אחרות של התראות, וכן כל צינורות יצירת התוכן.

כאשר משתמש מתקשר עם היישום או מפעיל בקשת תוכן ספציפית, היישום קולט את ההקשר הרלוונטי, כגון העדפות משתמש, אינטראקציות קודמות, או הנחיות ספציפיות. מידע הקשרי זה מוזן לתוך ה-LLM, יחד עם תבניות או הנחיות נחוצות, ומשמש ליצירת פלט טקסטואלי שאחרת היה צריך להיות מקודד מראש, מאוחסן במסד נתונים, או מיוצר אלגוריתמית.

התוכן המיוצר על ידי LLM יכול לקבל צורות שונות, כגון המלצות מותאמות אישית, תיאורי מוצרים דינמיים, תשובות דואר אלקטרוני מותאמות אישית, או אפילו מאמרים שלמים או פוסטים בבלוג. אחד השימושים הרדיקליים ביותר בתוכן זה שחידשתי לפני יותר משנה הוא יצירה דינמית של אלמנטי ממשק משתמש כמו תוויות טפסים, חלוניות הסבר, וסוגים אחרים של טקסט הסברתי.

התאמה אישית

אחד היתרונות המרכזיים של תבניות יצירת תוכן הקשרי הוא היכולת לספק חוויות מותאמות אישית מאוד למשתמשים. על ידי יצירת תוכן המבוסס על הקשר ספציפי למשתמש, תבניות אלה מאפשרות ליישומים להתאים תוכן לתחומי העניין, העדפות ואינטראקציות של משתמשים בודדים.

התאמה אישית היא הרבה מעבר להכנסה פשוטה של שם המשתמש לתוך תוכן גנרי. היא כוללת ניצול ההקשר העשיר הזמין לגבי כל משתמש כדי ליצור תוכן שמהדהד עם הצרכים והרצונות הספציפיים שלהם. הקשר זה יכול לכלול מגוון רחב של גורמים, כגון:

1. **מידע פרופיל משתמש:** ברמה הכללית ביותר של יישום טכניקה זו, ניתן להשתמש בנתונים דמוגרפיים, תחומי עניין, העדפות ותכונות פרופיל אחרות כדי ליצור תוכן שמתאים לרקע ולמאפיינים של המשתמש.

2. **נתוני התנהגות:** האינטראקציות הקודמות של המשתמש עם האפליקציה, כגון דפים שנצפו, קישורים שנלחצו או מוצרים שנרכשו, יכולות לספק תובנות חשובות לגבי ההתנהגות והעניין שלהם. ניתן להשתמש בנתונים אלה כדי ליצור הצעות תוכן המשקפות את דפוסי המעורבות שלהם וחוזות את צורכיהם העתידיים.
3. **גורמים הקשריים:** ההקשר הנוכחי של המשתמש, כגון מיקומם, המכשיר שלהם, שעת היום, ואפילו מזג האוויר, יכולים להשפיע על תהליך יצירת התוכן. לדוגמה, אפליקציית נסיעות עשויה להכיל עובד בינה מלאכותית שמסוגל ליצור המלצות מותאמות אישית בהתבסס על המיקום הנוכחי של המשתמש ותנאי מזג האוויר השוררים.

באמצעות ניצול גורמים הקשריים אלה, דפוסי יצירת תוכן הקשרי מאפשרים לאפליקציות לספק תוכן שמרגיש כאילו נתפר במיוחד עבור כל משתמש. לרמה זו של התאמה אישית יש מספר יתרונות משמעותיים:

1. **הגברת מעורבות:** תוכן מותאם אישית לוכד את תשומת הלב של המשתמשים ושומר על מעורבותם באפליקציה. כאשר משתמשים מרגישים שהתוכן רלוונטי ופונה ישירות לצרכיהם, הם נוטים יותר לבלות זמן רב יותר באינטראקציה עם האפליקציה ובחקירת תכונותיה.
2. **שיפור שביעות רצון המשתמש:** תוכן מותאם אישית מדגים שהאפליקציה מבינה ואכפת לה מהדרישות הייחודיות של המשתמש. על ידי אספקת תוכן מועיל, אינפורמטיבי ומותאם לתחומי העניין שלהם, האפליקציה יכולה להגביר את שביעות רצון המשתמש ולבנות קשר חזק יותר עם משתמשיה.
3. **שיעורי המרה גבוהים יותר:** בהקשר של אפליקציות מסחר אלקטרוני או שיווק, תוכן מותאם אישית יכול להשפיע משמעותית על שיעורי ההמרה. על ידי הצגת מוצרים, הצעות או המלצות המותאמות להעדפות ולהתנהגות של המשתמשים, האפליקציה יכולה להגדיל את הסבירות שמשתמשים יבצעו פעולות רצויות, כגון ביצוע רכישה או הרשמה לשירות.

פריון

דפוסי יצירת תוכן הקשרי יכולים להגביר משמעותית סוגים מסוימים של פריון על ידי הפחתת הצורך ביצירת תוכן ידנית ועריכה בתהליכים יצירתיים. באמצעות ניצול כוחם של מודלים

שפתיים גדולים, ניתן ליצור תוכן איכותי בקנה מידה גדול, תוך חיסכון בזמן ובמאמץ שיוצרי התוכן והמפתחים שלך היו צריכים להשקיע בעבודה ידנית מייגעת.

באופן מסורתי, יוצרי תוכן נדרשים לחקור, לכתוב, לערוך ולעצב תוכן כדי להבטיח שהוא עומד בדרישות האפליקציה ובציפיות המשתמשים. תהליך זה יכול להיות צורך זמן ומשאבים רבים, במיוחד ככל שהיקף התוכן גדל.

עם זאת, באמצעות תבניות יצירת תוכן הקשרי, ניתן לאוטמט במידה רבה את תהליך יצירת התוכן. מודלים לשוניים גדולים יכולים ליצור תוכן קוהרנטי, נכון מבחינה דקדוקית ורלוונטי להקשר בהתבסס על ההנחיות והקווים המנחים שסופקו. אוטומציה זו מציעה מספר יתרונות מבחינת פרודוקטיביות:

1. **הפחתת מאמץ ידני:** על ידי האצלת משימות יצירת תוכן למודלים לשוניים גדולים, יוצרי התוכן יכולים להתמקד במשימות ברמה גבוהה יותר כמו אסטרטגיית תוכן, העלאת רעיונות והבטחת איכות. הם יכולים לספק את ההקשר הנדרש, התבניות וההנחיות למודל הלשוני ולתת לו לטפל ביצירת התוכן עצמה. דבר זה מפחית את המאמץ הידני הנדרש לכתיבה ועריכה, ומאפשר ליוצרי התוכן להיות יותר פרודוקטיביים ויעילים.
2. **יצירת תוכן מהירה יותר:** מודלים לשוניים גדולים יכולים ליצור תוכן מהר הרבה יותר מכותבים אנושיים. עם ההנחיות והקווים המנחים הנכונים, מודל לשוני גדול יכול להפיק מספר פריטי תוכן תוך שניות או דקות. מהירות זו מאפשרת לאפליקציות ליצור תוכן בקצב מהיר בהרבה, תוך עמידה בדרישות המשתמשים והנוף הדיגיטלי המשתנה תדיר.

> האם יצירת תוכן מהירה יותר מובילה למצב של "טרגדיית המשאב המשותף" שבו האינטרנט טובע בתוכן שאף אחד לא קורא? לצערי, אני חושש שהתשובה היא כן.

3. **עקביות ואיכות:** מודלים לשוניים גדולים יכולים לתקן תוכן בקלות כך שיהיה עקבי בסגנון, טון ואיכות. בהינתן הנחיות ודוגמאות ברורות, סוגים מסוימים של אפליקציות (כלומר, חדר חדשות, יחסי ציבור וכו') יכולים להבטיח שהתוכן שנוצר על ידי בני אדם מתיישר עם קול המותג שלהם ועומד בסטנדרטים האיכותיים הרצויים. עקביות זו

מפחיתה את הצורך בעריכה ותיקונים נרחבים, וחוסכת זמן ומאמץ בתהליך יצירת התוכן.

4. **איטרציה ואופטימיזציה:** תבניות יצירת תוכן הקשרי מאפשרות איטרציה ואופטימיזציה מהירה של תוכן. על ידי התאמת ההנחיות, התבניות או הקווים המנחים שניתנים למודל הלשוני הגדול, האפליקציות שלכם יכולות במהירות ליצור וריאציות של תוכן ולבדוק גישות שונות באופן אוטומטי שמעולם לא היה אפשרי בעבר. תהליך איטרטיבי זה מאפשר ניסוי ושיפור מהיר יותר של אסטרטגיות תוכן, המוביל לתוכן יעיל ומעורר עניין יותר לאורך זמן. טכניקה מסוימת זו יכולה להיות משנה משחק מוחלט עבור אפליקציות כמו מסחר אלקטרוני שחיות או מתות על בסיס שיעורי נטישה ומעורבות

חשוב לציין כי בעוד שדפוסי יצירת תוכן הקשרי יכולים לשפר משמעותית את הפרודוקטיביות, הם אינם מבטלים לחלוטין את הצורך במעורבות אנושית. יוצרי תוכן ועורכים עדיין ממלאים תפקיד מכריע בהגדרת אסטרטגיית התוכן הכוללת, במתן הנחיות ל-LLM, ובהבטחת האיכות וההתאמה של התוכן המיוצר.

באמצעות אוטומציה של ההיבטים החזרתיים והזמן-תובעניים של יצירת התוכן, דפוסי יצירת תוכן הקשרי משחררים זמן ומשאבים אנושיים יקרי ערך שניתן להפנות למשימות בעלות ערך גבוה יותר. פרודוקטיביות מוגברת זו מאפשרת לך לספק תוכן מותאם אישית ומעורר יותר למשתמשים תוך כדי ייעול תהליכי יצירת התוכן.

איטרציה וניסוי מהירים

דפוסי יצירת תוכן הקשרי מאפשרים לך לבצע איטרציות ולנסות במהירות גרסאות שונות של תוכן, מה שמאפשר אופטימיזציה ושיפור מהירים יותר של אסטרטגיית התוכן שלך. אתה יכול ליצור מספר גרסאות של תוכן תוך שניות, פשוט על ידי התאמת ההקשר, התבניות או ההנחיות שניתנו למודל.

יכולת האיטרציה המהירה הזו מציעה מספר יתרונות מרכזיים:

1. **בדיקה ואופטימיזציה:** עם היכולת ליצור במהירות גרסאות תוכן, אתה יכול בקלות לבדוק גישות שונות ולמדוד את יעילותן. לדוגמה, אתה יכול ליצור מספר גרסאות

של תיאור מוצר או מסר שיווקי, כל אחד מותאם למגזר משתמשים ספציפי או הקשר מסוים. על ידי ניתוח מדדי מעורבות משתמשים, כגון שיעורי הקלקה או שיעורי המרה, אתה יכול לזהות את גרסאות התוכן היעילות ביותר ולייעל את אסטרטגיית התוכן שלך בהתאם.

2. **בדיקות A/B:** דפוסי יצירת תוכן הקשרי מאפשרים בדיקות A/B חלקות של תוכן. אתה יכול ליצור שתי גרסאות או יותר של תוכן ולהציג אותן באופן אקראי לקבוצות משתמשים שונות. על ידי השוואת הביצועים של כל גרסה, אתה יכול לקבוע איזה תוכן מתחבר הכי טוב לקהל היעד שלך. גישה מבוססת נתונים זו מאפשרת לך לקבל החלטות מושכלות ולשפר באופן מתמיד את התוכן שלך כדי למקסם את מעורבות המשתמשים ולהשיג את התוצאות הרצויות.
3. **ניסויי התאמה אישית:** איטרציה מהירה וניסוי הם בעלי ערך מיוחד כשמדובר בהתאמה אישית. עם דפוסי יצירת תוכן הקשרי, אתה יכול ליצור במהירות גרסאות תוכן מותאמות אישית המבוססות על מגזרי משתמשים שונים, העדפות או התנהגויות. על ידי ניסוי באסטרטגיות התאמה אישית שונות, אתה יכול לזהות את הגישות היעילות ביותר למעורבות משתמשים אינדיבידואלית ולאספקת חוויות מותאמות.
4. **הסתגלות למגמות משתנות:** היכולת לבצע איטרציות ולנסות במהירות מאפשרת לך להישאר זריז ולהסתגל למגמות משתנות והעדפות משתמשים. כאשר נושאים חדשים, מילות מפתח, או התנהגויות משתמשים מתפתחים, אתה יכול במהירות ליצור תוכן שמתיישר עם מגמות אלה. על ידי ניסוי ושיפור מתמיד של התוכן שלך, אתה יכול להישאר רלוונטי ולשמור על יתרון תחרותי בנוף הדיגיטלי המתפתח תמיד.
5. **ניסוי חסכוני:** ניסויי תוכן מסורתיים דורשים לרוב זמן ומשאבים משמעותיים, כאשר יוצרי התוכן נדרשים לפתח ולבדוק וריאציות שונות באופן ידני. עם זאת, באמצעות תבניות יצירת תוכן הקשרי, עלות הניסוי מופחתת משמעותית. מודלים שפתיים גדולים יכולים ליצור וריאציות תוכן במהירות ובקנה מידה גדול, מה שמאפשר לחקור מגוון רחב של רעיונות וגישות ללא עלויות משמעותיות.

כדי למקסם את התועלת מאיטרציה מהירה וניסויים, חשוב שתהיה מסגרת ניסויית מוגדרת היטב. מסגרת זו צריכה לכלול:

- מטרות והשערות ברורות לכל ניסוי

- מדדים ומנגנוני מעקב מתאימים למדידת ביצועי התוכן
- אסטרטגיות פילוח ומיקוד להבטחת הצגת וריאציות תוכן רלוונטיות למשתמשים הנכונים
- כלי ניתוח ודיווח להפקת תובנות מהנתונים הניסויים
- תהליך להטמעת לקחים ואופטימיזציות באסטרטגיית התוכן שלך

על ידי אימוץ איטרציה מהירה וניסויים, תוכל לשפר ולמטב את התוכן שלך באופן מתמיד, תוך הבטחה שהוא נשאר מעניין, רלוונטי ויעיל בהשגת מטרות היישום שלך. גישה זריזה זו ליצירת תוכן מאפשרת לך להישאר בחזית ולספק חוויות משתמש יוצאות דופן.

יכולת הרחבה ויעילות

ככל שיישומים גדלים והביקוש לתוכן מותאם אישית עולה, תבניות יצירת תוכן הקשרי מאפשרות הרחבה יעילה של יצירת התוכן. מודלים שפתיים גדולים יכולים ליצור תוכן למספר רב של משתמשים והקשרים בו-זמנית, ללא צורך בגידול פרופורציונלי במשאבי אנוש. יכולת הרחבה זו מאפשרת ליישומים לספק חוויות מותאמות אישית לבסיס משתמשים גדל מבלי להעמיס על יכולות יצירת התוכן שלהם.

שימו לב שניתן להשתמש ביצירת תוכן הקשרי באופן יעיל כדי לבצע בינאום של היישום שלכם "תוך כדי תנועה". למעשה, זה בדיוק מה שעשיתי באמצעות ה-Gem Instant18n שלי כדי לספק את Olympia ביותר מחצי תריסר שפות, למרות שאנחנו בני פחות משנה.

לוקליזציה מבוססת בינה מלאכותית

אם תרשו לי להתגאות לרגע, אני חושב שספריית Instant18n שלי ליישומי Rails היא דוגמה פורצת דרך לתבנית "יצירת תוכן הקשרי" בפעולה, המדגימה את הפוטנציאל המשנה של בינה מלאכותית בפיתוח יישומים. ספרייה זו מנצלת את כוחו של מודל השפה הגדול GPT של OpenAI כדי לחולל מהפכה באופן שבו מטופלים בינאום ולוקליזציה ביישומי Rails.

באופן מסורתי, בינאום יישום Rails כולל הגדרה ידנית של מפתחות תרגום ואספקת תרגומים מתאימים לכל שפה נתמכת. תהליך זה יכול להיות צורך זמן רב, אינטנסיבי במשאבים ונוטה לחוסר עקביות. עם זאת, עם ספריית Instant18n, הפרדיגמה של לוקליזציה מוגדרת מחדש לחלוטין.

באמצעות שילוב מודל שפה גדול, ה-gem בשם Instant18n מאפשר לייצר תרגומים באופן מיידי, בהתבסס על ההקשר והמשמעות של הטקסט. במקום להסתמך על מפתחות תרגום מוגדרים מראש ותרגומים סטטיים, ה-gem מתרגם טקסט באופן דינמי באמצעות כוח הבינה המלאכותית. גישה זו מציעה מספר יתרונות מרכזיים:

1. **לוקליזציה חלקה:** עם ה-gem של Instant18n, מפתחים אינם צריכים עוד להגדיר ולתחזק ידנית קבצי תרגום עבור כל שפה נתמכת. ה-gem מייצר באופן אוטומטי תרגומים בהתבסס על הטקסט המסופק והשפה המבוקשת, מה שהופך את תהליך הלוקליזציה לקל וחלק.
2. **דיוק הקשרי:** ניתן לספק לבינה המלאכותית מספיק הקשר כדי להבין את הניואנסים של הטקסט המתורגם. היא יכולה להתחשב בהקשר הסובב, בביטויים ובהתייחסויות תרבותיות כדי לייצר תרגומים מדויקים, טבעיים ומתאימים להקשר.
3. **תמיכה נרחבת בשפות:** ה-gem של Instant18n מנצל את הידע והיכולות הלשוניות הנרחבות של GPT, ומאפשר תרגום למגוון רחב של שפות. משפות נפוצות כמו ספרדית וצרפתית ועד שפות נדירות או בדיוניות כמו קלינגונית ואלפית, ה-gem יכול לטפל במגוון רחב של דרישות תרגום.
4. **גמישות ויצירתיות:** ה-gem חורג מעבר לתרגומי שפה מסורתיים ומאפשר אפשרויות לוקליזציה יצירתיות ולא שגרתיות. מפתחים יכולים לתרגם טקסט לסגנונות שונים, ניבים, ואפילו שפות בדיוניות, מה שפותח אפשרויות חדשות לחוויות משתמש ייחודיות ותוכן מעניין.
5. **אופטימיזציה של ביצועים:** ה-gem של Instant18n כולל מנגנוני מטמון לשיפור הביצועים והפחתת העומס של תרגומים חוזרים. טקסט מתורגם נשמר במטמון, מה שמאפשר לבקשות עוקבות לאותו תרגום להיות מוגשות במהירות ללא צורך בקריאות API מיותרות.

ה-gem של Instant18n מדגים את כוחו של דפוס "יצירת תוכן הקשרי" על ידי ניצול בינה

מלאכותית ליצירת תוכן מקומי באופן דינמי. הוא מראה כיצד ניתן לשלב בינה מלאכותית בפונקציונליות הליבה של יישום ,Rails ומשנה את האופן שבו מפתחים מתייחסים לבינאום ולוקליזציה.

על ידי ביטול הצורך בניהול תרגומים ידני ואפשור תרגומים מיידיים המבוססים על הקשר, ה-gem של Instant18n חוסך למפתחים זמן ומאמץ משמעותיים. הוא מאפשר להם להתמקד בבניית תכונות הליבה של היישום שלהם תוך הבטחה שהיבט הלוקליזציה מטופל באופן חלק ומדויק.

חשיבות בדיקות המשתמשים והמשוב

לבסוף, חשוב תמיד לזכור את חשיבות בדיקות המשתמשים והמשוב. חיוני לוודא שיצירת התוכן ההקשרי עומדת בציפיות המשתמשים ומתיישרת עם מטרות היישום. יש להמשיך ולשפר את התוכן המיוצר בהתבסס על תובנות משתמשים וניתוח נתונים. אם אתם מייצרים תוכן דינמי בהיקף גדול שבלתי אפשרי לאמת באופן ידני על ידי הצוות שלכם, שקלו להוסיף מנגנוני משוב המאפשרים למשתמשים לדווח על תוכן מוזר או שגוי, יחד עם הסבר מדוע. המשוב היקר הזה יכול אפילו להיות מועבר לעובד בינה מלאכותית שתפקידו לבצע התאמות ברכיב שייצר את התוכן!

ממשק משתמש גנרטיבי

קשב הוא משאב כה יקר בימינו, עד שמעורבות משתמש אפקטיבית דורשת כעת חוויות תוכנה שאינן רק חלקות ואינטואיטיביות, אלא גם מותאמות אישית לצרכים, להעדפות ולהקשרים של כל משתמש. כתוצאה מכך, מעצבים ומפתחים ניצבים בפני האתגר של יצירת ממשקי משתמש שיכולים להסתגל ולהתאים לדרישות הייחודיות של כל משתמש בקנה מידה רחב.

ממשק משתמש גנרטיבי (GenUI) הוא גישה מהפכנית באמת לעיצוב ממשק משתמש המנצלת את כוחם של מודלים שפתיים גדולים (LLMs) כדי ליצור חוויות משתמש מותאמות אישית ודינמיות תוך כדי תנועה. רציתי לוודא שאתן לכם לפחות מבוא בסיסי ל-GenUI בספר זה, כי אני מאמין שזו אחת ההזדמנויות הפתוחות והחדשות ביותר שקיימות כיום בתחום של עיצוב יישומים ומסגרות עבודה. אני משוכנע שעשרות או יותר פרויקטים מסחריים ובקוד

פתוח מוצלחים יצוצו בנישה הספציפית הזו.

בליבו, GenUI משלב את העקרונות של יצירת תוכן הקשרי עם טכניקות בינה מלאכותית מתקדמות כדי ליצור אלמנטים של ממשק משתמש, כגון טקסט, תמונות ופריסות, באופן דינמי על בסיס הבנה עמוקה של ההקשר, ההעדפות והמטרות של המשתמש. GenUI מאפשר למעצבים ומפתחים ליצור ממשקים שמסתגלים ומתפתחים בתגובה לאינטראקציות של המשתמש, ומספקים רמה של התאמה אישית שלא הייתה אפשרית בעבר.

GenUI מייצג שינוי יסודי באופן שבו אנו ניגשים לעיצוב ממשק משתמש. במקום לעצב עבור ההמונים, GenUI מאפשר לנו לעצב עבור הפרט. תוכן וממשקים מותאמים אישית יש להם את הפוטנציאל ליצור חוויות משתמש שמהדהדות עם כל משתמש ברמה עמוקה יותר, ומגבירות מעורבות, שביעות רצון ונאמנות.

כטכניקה חדשנית, המעבר ל-GenUI מלא באתגרים קונספטואליים ומעשיים. שילוב בינה מלאכותית בתהליך העיצוב, הבטחה שהממשקים המיוצרים אינם רק מותאמים אישית אלא גם שימושיים, נגישים ומתואמים עם המותג וחוויית המשתמש הכוללת - כל אלה הם אתגרים שהופכים את GenUI למרדף עבור המעטים, לא הרבים. בנוסף, המעורבות של בינה מלאכותית מעלה שאלות לגבי פרטיות מידע, שקיפות, ואולי אפילו השלכות אתיות

למרות האתגרים, חוויות מותאמות אישית בקנה מידה רחב יכולות לשנות לחלוטין את האופן שבו אנו מתקשרים עם מוצרים ושירותים דיגיטליים. הדבר פותח אפשרויות ליצירת ממשקים מכילים ונגישים המותאמים לצרכים המגוונים של המשתמשים, ללא קשר ליכולותיהם, הרקע שלהם או העדפותיהם.

בפרק זה נחקור את מושג ה-GenUI, תוך בחינת מספר מאפיינים מגדירים, יתרונות מרכזיים ואתגרים פוטנציאליים. נתחיל בבחינת הצורה הבסיסית והנגישה ביותר של GenUI: יצירת טקסט עבור ממשקי משתמש המתוכננים ומיושמים באופן מסורתי.

יצירת טקסט עבור ממשקי משתמש

אלמנטים טקסטואליים הקיימים בממשק המשתמש הקבוע של האפליקציה שלכם, כמו תוויות טפסים, חלוניות הסבר וטקסט הסברתי, בדרך כלל מקודדים ישירות בתבניות או ברכיבי ממשק המשתמש, ומספקים חוויה עקבית אך גנרית לכל המשתמשים. באמצעות דפוסי

יצירת תוכן הקשרי, ניתן להפוך את האלמנטים הסטטיים הללו לרכיבים דינמיים, מודעי הקשר ומותאמים אישית.

טפסים מותאמים אישית

טפסים הם חלק בלתי נפרד מאפליקציות אינטרנט ומובייל, ומשמשים כאמצעי העיקרי לאיסוף קלט מהמשתמש. עם זאת, טפסים מסורתיים מציגים לרוב חוויה גנרית ולא אישית, עם תוויות ושדות סטנדרטיים שלא תמיד מתאימים להקשר או לצרכים הספציפיים של המשתמש. משתמשים נוטים יותר להשלים טפסים שמרגישים מותאמים לצרכים ולהעדפות שלהם, מה שמוביל לשיעורי המרה גבוהים יותר ושביעות רצון משתמשים גבוהה יותר.

עם זאת, חשוב למצוא את האיזון הנכון בין התאמה אישית לבין עקביות. בעוד שהתאמת טפסים למשתמשים ספציפיים יכולה להיות מועילה, חיוני לשמור על רמה של היכרות וצפיות. משתמשים צריכים עדיין להיות מסוגלים לזהות ולנווט בטפסים בקלות, גם עם אלמנטים מותאמים אישית.

הנה כמה רעיונות לטפסים מותאמים אישית להשראה:

הצעות שדה הקשריות

GenUI יכול לנתח את האינטראקציות הקודמות של המשתמש, העדפותיו ונתוניו כדי לספק הצעות שדה חכמות כתחזיות. למשל, אם המשתמש הזין בעבר את כתובת המשלוח שלו, הטופס יכול למלא אוטומטית את השדות הרלוונטיים עם המידע השמור שלו. זה לא רק חוסך זמן אלא גם מדגים שהאפליקציה מבינה וזוכרת את העדפות המשתמש.

רגע, האם טכניקה זו אינה משהו שניתן לעשות ללא שימוש בבינה מלאכותית? כמובן, אבל היופי בהפעלת פונקציונליות מסוג זה באמצעות בינה מלאכותית הוא כפול: 1) כמה קל ליישם זאת ו-2) כמה עמיד זה יכול להיות כאשר ממשק המשתמש שלך משתנה ומתפתח לאורך זמן.

בואו ניצור במהירות שירות עבור מערכת טיפול בהזמנות תיאורטית שלנו, שמנסה למלא באופן יזום את כתובת המשלוח הנכונה עבור המשתמש.

```
class OrderShippingAddressSubscriber
  include Raix::ChatCompletion

  attr_accessor :order

  delegate :customer, to: :order

  DIRECTIVE = "the Given assistant. processing order smart a are You
address shipping likely most the guess ,history order customer's
order. current the for  "

  def order_created(order)
    return unless order.pending? && order.shipping_address.blank?

    self.order = order

    transcript.clear
    transcript << { system: DIRECTIVE }
    transcript << { user: " History: Order#{order_history.to_json}" }
    transcript << { user: " Order: Current#{order.to_json}" }

    response = chat_completion
    apply_predicted_shipping_address(order, response)
  end

  private

  def apply_predicted_shipping_address(order, response)
    response... the from address shipping the extract #
    fields address the of update live of sort some there's assume ...and #
    order.update(shipping_address:)
  end

  def order_history
    customer.orders.successful.limit(100).map do |order|
      {
        date: order.date,
        description: order.description,
        shipping_address: order.shipping_address
      }
    end
  end
```

```
end
```

הדוגמה הזו מאוד מפושטת, אך אמורה לעבוד ברוב המקרים. הרעיון הוא לתת לבינה המלאכותית לנחש באותו אופן שבו אדם היה מנחש. כדי להבהיר למה אני מתכוון, הבה נבחן כמה נתוני דוגמה:

```
Order History:
[
 {"date": "2024-01-03", "description": "garden soil mix",
  "shipping_address": "123 Country Lane, Rural Town"},
 {"date": "2024-01-15", "description": "hardcover fiction novels",
  "shipping_address": "456 City Apt, Metroville"},
 {"date": "2024-01-22", "description": "baby diapers", "shipping_address":
  "789 Suburb St, Quietville"},
 {"date": "2024-02-01", "description": "organic vegetables",
  "shipping_address": "123 Country Lane, Rural Town"},
 {"date": "2024-02-17", "description": "mystery thriller book set",
  "shipping_address": "456 City Apt, Metroville"},
 {"date": "2024-02-25", "description": "baby wipes",
  "shipping_address": "789 Suburb St, Quietville"},
 {"date": "2024-03-05", "description": "flower seeds",
  "shipping_address": "123 Country Lane, Rural Town"},
 {"date": "2024-03-20", "description": "biographies",
  "shipping_address": "456 City Apt, Metroville"},
 {"date": "2024-03-30", "description": "baby formula",
  "shipping_address": "789 Suburb St, Quietville"},
 {"date": "2024-04-12", "description": "lawn fertilizer",
  "shipping_address": "123 Country Lane, Rural Town"},
 {"date": "2024-04-22", "description": "science fiction novels",
  "shipping_address": "456 City Apt, Metroville"},
 {"date": "2024-05-02", "description": "infant toys",
  "shipping_address": "789 Suburb St, Quietville"},
 {"date": "2024-05-14", "description": "outdoor grill",
  "shipping_address": "123 Country Lane, Rural Town"},
 {"date": "2024-05-29", "description": "literary classics",
 "shipping_address": "456 City Apt, Metroville"},
 {"date": "2024-06-11", "description": "baby clothes",
  "shipping_address": "789 Suburb St, Quietville"},
 {"date": "2024-07-01", "description": "watering can",
  "shipping_address": "123 Country Lane, Rural Town"},
```

```
  {"date": "2024-07-18", "description": "non-fiction essays",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-07-28", "description": "baby bath items",
  "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-08-09", "description": "herb garden kit",
  "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-08-24", "description": "children's books",
  "shipping_address": "456 City Apt, Metroville"}
]
```

האם שמת לב לדפוס בנתונים? אני מבטיח לך שזה חומר קל עבור מודל שפה גדול. כדי להדגים זאת, בוא נשאל את 4GPT- מהי כתובת המשלוח הסבירה ביותר עבור "מדחום".

```
From the order history you've provided, it looks like the purchases are
generally clustered into three main types based on the shipping addresses:

123 Country Lane, Rural Town - This address often orders garden and
outdoor-related items like soil mix, vegetables, flower seeds, lawn
fertilizer, an outdoor grill, a watering can, and an herb garden kit.

456 City Apt, Metroville - This address tends to order various types of
books, indicating an interest in reading across genres like fiction,
thrillers, biographies, science fiction, literary classics, non-fiction
essays, and children's books.

789 Suburb St, Quietville - The purchases for this address are consistently
baby-related products, such as diapers, wipes, baby formula, infant toys,
baby clothes, and baby bath items.

Given these patterns, a digital thermometer could potentially be a fit for
any of the addresses based on different needs: outdoor temperature tracking,
a tool for a passionate reader interested in science gadgets, or a health
tool for a baby. However, considering its common use in monitoring health,
especially for babies, it's most likely to be purchased by someone who's
already buying baby care products.

Therefore, the most likely shipping address for a digital thermometer, based
on the purchasing patterns, would be 789 Suburb St, Quietville.
```

אם אתה חושב לעצמך שזו הגזמה להשתמש במודל יקר כמו 4GPT- למשימה הזו, אתה צודק!

ניסיתי את אותו פרומפט על Instruct 7B Mistral והוא הפיק את התגובה הבאה במהירות של 75 טוקנים לשנייה, ובעלות זניחה של 0.000218 דולר אמריקאי.

```
Based on the given order history, it appears that the customer has received
orders for baby-related items (diapers, wipes, formula, clothes, bath items)
at the address "789 Suburb St, Quietville" and garden-related items (soil,
seeds, watering can, herb garden kit) at the address "123 Country Lane, Rural
Town". The customer has also received books (fiction, non-fiction, literary
classics, mystery-thriller, science fiction, biographies, children's books)
at the address "456 City Apt, Metroville".

Given that the new order is for a "thermometer", which doesn't seem to fit
into any of the categories that the customer has previously ordered, it's
hard to accurately predict the shipping address based solely on the order
history. However, given the 50%-50% split between baby-related and
garden-related items, it could somewhat lean towards the Baby-related items
address ("789 Suburb St, Quietville"). But remember, this is an assumption
and cannot be definitively confirmed without more context or information.
```

האם השקעת המשאבים והעלות של טכניקה זו שווים את המאמץ כדי להפוך את חוויית התשלום ליותר קסומה? עבור הרבה קמעונאים מקוונים, בהחלט כן. ולפי המראה, עלות החישוב של בינה מלאכותית רק תמשיך לרדת, במיוחד עבור ספקי אירוח מודלים בקוד פתוח במרוץ לתחתית.

השתמש ב-תבנית הנחיה ו-קלט/פלט מובנה יחד עם תיחום תגובות כדי לייעל את סוג זה של השלמת צ'אט.

סידור שדות אדפטיבי

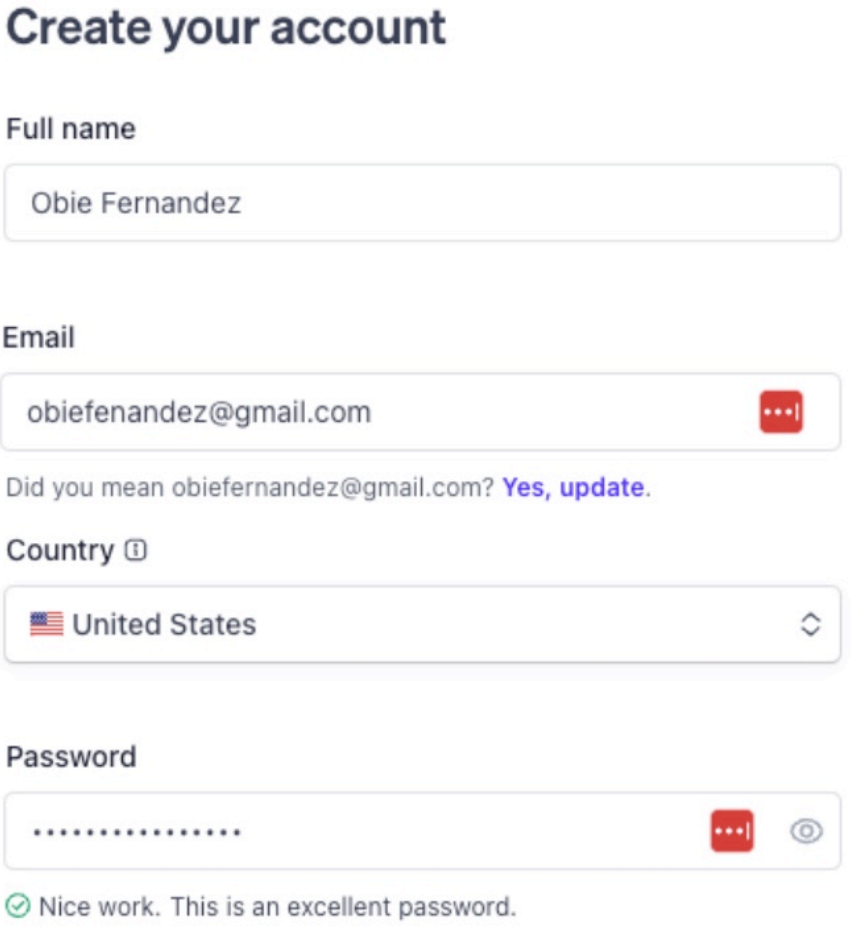
הסדר שבו מוצגים שדות הטופס יכול להשפיע משמעותית על חווית המשתמש ושיעורי ההשלמה. עם GenUI, ניתן להתאים דינמית את סדר השדות בהתבסס על ההקשר של המשתמש וחשיבות כל שדה. לדוגמה, אם המשתמש ממלא טופס הרשמה לאפליקציית כושר, הטופס יכול לתעדף שדות הקשורים למטרות הכושר והעדפות שלהם, מה שהופך את התהליך לרלוונטי ומעורב יותר.

טקסט משני מותאם אישית

טקסט ההוראות, הודעות שגיאה וטקסטים משניים אחרים המשויכים לטפסים יכולים גם הם להיות מותאמים אישית באמצעות GenUI. במקום להציג הודעות שגיאה כלליות כמו "כתובת אימייל לא תקינה", ניתן ליצור הודעות מועילות והקשריות יותר כמו "אנא הזן כתובת אימייל תקינה כדי לקבל את אישור ההזמנה שלך". נגיעות אישיות אלה יכולות להפוך את חווית הטופס לידידותית יותר למשתמש ופחות מתסכלת.

אימות מותאם אישית

בדומה לטקסט המשני המותאם אישית, ניתן להשתמש בבינה מלאכותית כדי לאמת את הטופס בדרכים שנראות קסומות. דמיינו שמאפשרים לבינה מלאכותית לאמת טופס פרופיל משתמש, תוך חיפוש טעויות פוטנציאליות ברמה סמנטית.



איור 12.9. האם אתם יכולים לזהות את האימות הסמנטי המתרחש?

חשיפה הדרגתית

GenUI יכול לזהות באופן חכם אילו שדות טופס הם חיוניים בהתבסס על ההקשר של המשתמש ולחשוף בהדרגה שדות נוספים לפי הצורך. טכניקת החשיפה ההדרגתית הזו עוזרת להפחית את העומס הקוגניטיבי והופכת את תהליך מילוי הטופס לקל יותר לניהול. למשל, אם משתמש נרשם למנוי בסיסי, הטופס יכול להציג תחילה רק את השדות החיוניים,

וככל שהמשתמש מתקדם או בוחר באפשרויות ספציפיות, שדות רלוונטיים נוספים יכולים להופיע באופן דינמי.

טקסט הסברים מודע הקשר

חלוניות עזרה משמשות לעתים קרובות כדי לספק מידע נוסף או הדרכה למשתמשים כאשר הם מרחפים מעל או מתקשרים עם אלמנטים ספציפיים. באמצעות גישת "יצירת תוכן תלוי הקשר", ניתן ליצור חלוניות עזרה המסתגלות להקשר של המשתמש ומספקות מידע רלוונטי. למשל, אם משתמש חוקר תכונה מורכבת, חלונית העזרה יכולה להציע טיפים מותאמים אישית או דוגמאות בהתבסס על האינטראקציות הקודמות שלו או רמת המיומנות שלו.

טקסט הסברים, כגון הוראות, תיאורים או הודעות עזרה, יכול להיווצר באופן דינמי בהתבסס על ההקשר של המשתמש. במקום להציג הסברים כלליים, ניתן להשתמש במודלי שפה גדולים כדי ליצור טקסט המותאם לצרכים או לשאלות הספציפיות של המשתמש. לדוגמה, אם משתמש מתקשה בשלב מסוים בתהליך, טקסט ההסבר יכול לספק הדרכה מותאמת אישית או טיפים לפתרון בעיות.

טקסט ממשק מתייחס לקטעי טקסט קטנים המנחים משתמשים דרך היישום שלך, כמו תוויות כפתורים, הודעות שגיאה או הודעות אישור. על ידי יישום גישת יצירת תוכן תלוי הקשר לטקסט ממשק, ניתן ליצור ממשק משתמש מסתגל המגיב לפעולות המשתמש ומספק טקסט רלוונטי ומועיל. למשל, אם משתמש עומד לבצע פעולה קריטית, הודעת האישור יכולה להיווצר באופן דינמי כדי לספק מסר ברור ומותאם אישית.

טקסט הסברים מותאם אישית וחלוניות עזרה יכולים לשפר משמעותית את תהליך החניכה הראשונית עבור משתמשים חדשים. על ידי מתן הדרכה ודוגמאות תלויות הקשר, ניתן לעזור למשתמשים להבין ולנווט ביישום במהירות, להפחית את עקומת הלמידה ולהגדיל את האימוץ.

רכיבי ממשק דינמיים ומודעי הקשר יכולים גם לגרום ליישום להרגיש יותר אינטואיטיבי ומעורר עניין. סביר יותר שמשתמשים יתקשרו עם ויחקרו תכונות כאשר הטקסט הנלווה מותאם לצרכים ולתחומי העניין הספציפיים שלהם.

עד כה סקרנו רעיונות לשיפור פרדיגמות ממשק משתמש קיימות באמצעות בינה מלאכותית, אבל מה לגבי חשיבה מחדש על האופן שבו ממשקי משתמש מתוכננים ומיושמים בצורה רדיקלית יותר?

הגדרת ממשק משתמש גנרטיבי

בניגוד לעיצוב ממשק משתמש מסורתי, בו מעצבים יוצרים ממשקים קבועים וסטטיים, ממשק משתמש גנרטיבי מרמז על עתיד בו התוכנה שלנו מתהדרת בחוויות גמישות ומותאמות אישית שיכולות להתפתח ולהסתגל בזמן אמת. בכל פעם שאנחנו משתמשים בממשק שיחה מונע בינה מלאכותית, אנחנו מאפשרים לבינה המלאכותית להסתגל לצרכים הספציפיים של המשתמש. ממשק משתמש גנרטיבי לוקח את הדברים צעד קדימה על ידי יישום רמת ההסתגלות הזו לממשק החזותי של התוכנה.

הסיבה שאפשר לשחק עם רעיונות של ממשק משתמש גנרטיבי כיום היא שמודלי שפה גדולים. כבר מבינים תכנות והידע הבסיסי שלהם כולל טכנולוגיות ומסגרות עבודה של ממשק משתמש השאלה היא האם ניתן להשתמש במודלי שפה גדולים כדי ליצור אלמנטים של ממשק משתמש, כגון טקסט, תמונות, פריסות ואפילו ממשקים שלמים, המותאמים לכל משתמש באופן אישי. ניתן להנחות את המודל להתחשב בגורמים שונים, כמו אינטראקציות קודמות של המשתמש, העדפות מוצהרות, מידע דמוגרפי וההקשר הנוכחי של השימוש, כדי ליצור ממשקים מותאמים אישית ורלוונטיים ביותר.

ממשק משתמש גנרטיבי נבדל מעיצוב ממשק משתמש מסורתי במספר דרכים מרכזיות:

1. **דינמי ומסתגל:** עיצוב ממשק משתמש מסורתי כולל יצירת ממשקים קבועים וסטטיים שנשארים זהים עבור כל המשתמשים. לעומת זאת, ממשק משתמש גנרטיבי מאפשר ממשקים שיכולים להסתגל ולהשתנות באופן דינמי בהתבסס על צרכי המשתמש וההקשר. משמעות הדבר היא שאותו יישום יכול להציג ממשקים שונים למשתמשים שונים או אפילו לאותו משתמש במצבים שונים.
2. **התאמה אישית בקנה מידה רחב:** בעיצוב מסורתי, יצירת חוויות מותאמות אישית לכל משתמש היא לעתים קרובות לא מעשית בשל הזמן והמשאבים הנדרשים. ממשק משתמש גנרטיבי, לעומת זאת, מאפשר התאמה אישית בקנה מידה רחב. באמצעות

ניצול בינה מלאכותית, מעצבים יכולים ליצור ממשקים המסתגלים אוטומטית לצרכים ולהעדפות הייחודיים של כל משתמש, מבלי לצורך לעצב ולפתח ממשקים נפרדים לכל מגזר משתמשים.

3. **התמקדות בתוצאות:** עיצוב ממשק משתמש מסורתי מתמקד לעתים קרובות ביצירת ממשקים מושכים ופונקציונליים מבחינה חזותית. בעוד שהיבטים אלה עדיין חשובים בממשק משתמש גנרטיבי, המיקוד העיקרי עובר להשגת תוצאות רצויות למשתמש. ממשק משתמש גנרטיבי שואף ליצור ממשקים שמותאמים למטרות ולמשימות הספציפיות של כל משתמש, תוך מתן עדיפות לשימושיות ויעילות על פני שיקולים אסתטיים בלבד.
4. **למידה ושיפור מתמשכים:** מערכות ממשק משתמש גנרטיבי יכולות ללמוד ולהשתפר באופן מתמיד לאורך זמן בהתבסס על אינטראקציות ומשוב של משתמשים. כאשר משתמשים מתקשרים עם הממשקים שנוצרו, מודלי הבינה המלאכותית יכולים לאסוף נתונים על התנהגות משתמשים, העדפות ותוצאות, ולהשתמש במידע זה כדי לשפר ולמטב יצירת ממשקים עתידית. תהליך הלמידה האיטרטיבי הזה מאפשר למערכות ממשק משתמש גנרטיבי להיות יעילות יותר ויותר במילוי צרכי המשתמש לאורך זמן.

חשוב לציין כי GenUI שונה מכלי עיצוב בסיוע בינה מלאכותית, כגון אלה המספקים הצעות או מאתמטים משימות עיצוב מסוימות. בעוד שכלים אלה יכולים לעזור בייעול תהליך העיצוב, הם עדיין מסתמכים על מעצבים לקבלת החלטות סופיות וליצירת ממשקים סטטיים. מאידך, GenUI מערב את מערכת הבינה המלאכותית בתפקיד פעיל יותר ביצירה והתאמה בפועל של ממשקים על בסיס נתוני משתמש והקשר.

GenUI מייצג שינוי משמעותי באופן שבו אנו ניגשים לעיצוב ממשק משתמש, תוך התרחקות מפתרונות אחידים ומעבר לחוויות מותאמות אישית ואדפטיביות. באמצעות ניצול כוחה של הבינה המלאכותית, ל-GenUI יש את הפוטנציאל למהפך באופן שבו אנו מתקשרים עם מוצרים ושירותים דיגיטליים, תוך יצירת ממשקים שהם יותר אינטואיטיביים, מעורבים ויעילים עבור כל משתמש בנפרד.

דוגמה

כדי להמחיש את הרעיון של GenUI, הבה נתבונן באפליקציית כושר היפותטית בשם "FitAI". אפליקציה זו שואפת לספק תוכניות אימון ועצות תזונה מותאמות אישית למשתמשים בהתבסס על המטרות האישיות שלהם, רמות הכושר והעדפותיהם.

בגישת עיצוב ממשק משתמש מסורתית, ל-FitAI היה יכול להיות מערך קבוע של מסכים ואלמנטים שזהים לכל המשתמשים. עם זאת, עם GenUI, ממשק האפליקציה יכול להסתגל דינמית לצרכים ולהקשר הייחודיים של כל משתמש.

גישה זו היא מעט קשה לדמיון ליישום בשנת 2024 וייתכן שאפילו אין לה החזר השקעה מספק, אך היא אפשרית.

הנה כיצד זה עשוי לעבוד:

1. **תהליך ההצטרפות:**

- במקום שאלון סטנדרטי, FitAI משתמש בבינה מלאכותית שיחתית לאיסוף מידע על מטרות המשתמש, רמת הכושר הנוכחית והעדפותיו.
- בהתבסס על האינטראקציה הראשונית הזו, הבינה המלאכותית מייצרת פריסת לוח מחוונים מותאמת אישית, המדגישה את התכונות והמידע הרלוונטיים ביותר למטרות המשתמש.
- טכנולוגיית הבינה המלאכותית הנוכחית עשויה להשתמש באוסף של רכיבי מסך העומדים לרשותה ליצירת לוח המחוונים המותאם אישית.
- טכנולוגיית בינה מלאכותית עתידית עשויה לקחת על עצמה את תפקיד מעצב ממשק המשתמש המנוסה וליצור את לוח המחוונים מאפס.

2. **מתכנן אימונים:**

- ממשק מתכנן האימונים מותאם על ידי הבינה המלאכותית כדי להתאים ספציפית לרמת הניסיון של המשתמש והציוד הזמין.
- עבור מתחיל ללא ציוד, הוא עשוי להציג תרגילי משקל גוף פשוטים עם הוראות מפורטות וסרטונים.

- עבור משתמש מתקדם עם גישה לחדר כושר, הוא יכול להציג שגרות מורכבות יותר עם פחות תוכן הסברתי.
- תוכן מתכנן האימונים אינו פשוט מסונן מתוך קבוצה גדולה יותר. הוא יכול להיות מיוצר תוך כדי תנועה בהתבסס על מאגר ידע שנשאל עם הקשר הכולל את כל מה שידוע על המשתמש.

3. **מעקב התקדמות:**

- ממשק מעקב ההתקדמות מתפתח בהתאם ליעדי המשתמש ודפוסי המעורבות שלו.
- אם המשתמש מתמקד בעיקר בירידה במשקל, הממשק עשוי להציג באופן בולט גרף מגמת משקל ונתוני שריפת קלוריות.
- עבור משתמש שבונה שרירים, הוא עשוי להדגיש עלייה בכוח ושינויים בהרכב הגוף.
- הבינה המלאכותית יכולה להתאים חלק זה של האפליקציה להתקדמות בפועל של המשתמש. אם ההתקדמות נעצרת לתקופה מסוימת, האפליקציה יכולה לעבור למצב שבו היא מנסה לשכנע את המשתמש לחשוף את הסיבות לנסיגה, כדי למתן אותן.

4. **ייעוץ תזונתי:**

- מדור התזונה מתאים את עצמו להעדפות והגבלות התזונתיות של המשתמש.
- עבור משתמש טבעוני, הוא עשוי להציג הצעות לארוחות מהצומח ומקורות חלבון.
- עבור משתמש עם רגישות לגלוטן, הוא יסנן אוטומטית מזונות המכילים גלוטן מההמלצות.
- שוב, התוכן אינו נשאב מתוך קבוצה עצומה של נתוני ארוחות המתאימה לכל המשתמשים, אלא מסונתז ממאגר ידע המכיל מידע הניתן להתאמה בהתבסס על המצב והאילוצים הספציפיים של המשתמש.
- לדוגמה, מתכונים נוצרים עם מפרטי רכיבים התואמים את הצרכים הקלוריים המשתנים תדיר של המשתמש ככל שרמת הכושר ונתוני הגוף שלו מתפתחים.

5. **אלמנטים מוטיבציוניים:**

- התוכן המוטיבציוני וההתראות של האפליקציה מותאמים אישית בהתבסס על סוג האישיות של המשתמש והתגובה שלו לאסטרטגיות מוטיבציה שונות.
- חלק מהמשתמשים עשויים לקבל הודעות מעודדות, בעוד אחרים מקבלים משוב מבוסס נתונים.

בדוגמה זו, GenUI מאפשר ל-FitAI ליצור חוויה מותאמת אישית מאוד לכל משתמש, מה שעשוי להגביר את המעורבות, שביעות הרצון והסיכוי להשגת יעדי הכושר. אלמנטי הממשק, התוכן, ואפילו ה"אישיות" של האפליקציה מסתגלים כדי לשרת בצורה הטובה ביותר את הצרכים וההעדפות של כל משתמש.

המעבר לתכנון מוכוון תוצאות

GenUI מייצג שינוי יסודי בגישה לעיצוב ממשק משתמש!, במעבר מהתמקדות ביצירת אלמנטי ממשק ספציפיים לגישה הוליסטית יותר, מוכוונת תוצאות. לשינוי זה יש מספר השלכות חשובות:

1. **התמקדות ביעדי משתמש:**

- מעצבים יצטרכו לחשוב בצורה עמוקה יותר על יעדי משתמש ותוצאות רצויות במקום על רכיבי ממשק ספציפיים.
- הדגש יהיה על יצירת מערכות שיכולות לייצר ממשקים המסייעים למשתמשים להשיג את מטרותיהם ביעילות ואפקטיביות.
- מסגרות עבודה חדשות של ממשק משתמש יצוצו שיעניקו למעצבים מבוססי בינה מלאכותית את הכלים הנדרשים כדי לייצר חוויות משתמש תוך כדי תנועה ו_מאפס_ במקום על בסיס מפרטי מסך מוגדרים מראש.

2. **תפקיד המעצבים המשתנה:**

- מעצבים יעברו מיצירת פריסות קבועות להגדרת כללים, אילוצים והנחיות שמערכות בינה מלאכותית יצטרכו לעקוב אחריהם בעת יצירת ממשקים.

- הם יצטרכו לפתח מיומנויות בתחומים כמו ניתוח נתונים, הנדסת הנחיות בינה מלאכותית, וחשיבה מערכתית כדי להנחות ביעילות מערכות ממשק משתמש גנרטיבי.

3. **חשיבות מחקר המשתמשים:**

- מחקר משתמשים הופך לקריטי אף יותר בהקשר של ממשק משתמש גנרטיבי, כאשר מעצבים צריכים להבין לא רק את העדפות המשתמשים, אלא גם כיצד העדפות וצרכים אלה משתנים בהקשרים שונים.
- בדיקות משתמשים מתמשכות ומעגלי משוב יהיו חיוניים לשיפור וטיוב יכולת הבינה המלאכותית ליצור ממשקים יעילים.

4. **עיצוב עבור שונות:**

- במקום ליצור ממשק "מושלם" אחד, מעצבים יצטרכו לשקול מספר וריאציות אפשריות ולהבטיח שהמערכת יכולה ליצור ממשקים מתאימים לצרכי משתמשים מגוונים.
- זה כולל עיצוב עבור מקרי קצה והבטחה שהממשקים שנוצרים שומרים על שימושיות ונגישות בתצורות שונות.
- בידול מוצרים מקבל ממדים חדשים הכוללים נקודות מבט שונות על פסיכולוגיית משתמשים וניצול מערכי נתונים וידע ייחודיים שאינם זמינים למתחרים.

אתגרים ושיקולים

בעוד שממשק משתמש גנרטיבי מציע אפשרויות מרגשות, הוא מציג גם מספר אתגרים ושיקולים:

1. **מגבלות טכניות:**

- טכנולוגיית הבינה המלאכותית הנוכחית, למרות התקדמותה, עדיין מוגבלת בהבנת כוונות משתמש מורכבות וביצירת ממשקים מודעי הקשר באמת.

- בעיות ביצועים הקשורות ליצירה בזמן אמת של רכיבי ממשק, במיוחד במכשירים פחות חזקים.

2. **דרישות נתונים:**

- בהתאם למקרה השימוש, מערכות ממשק משתמש גנרטיבי יעילות עשויות לדרוש כמויות משמעותיות של נתוני משתמש כדי ליצור ממשקים מותאמים אישית.
- האתגרים באיסוף אתי של נתוני משתמש אמיתיים מעלים חששות לגבי פרטיות ואבטחת מידע, כמו גם הטיות אפשריות בנתונים המשמשים לאימון מודלים של ממשק משתמש גנרטיבי.

3. **שימושיות ועקביות:**

- לפחות עד שהפרקטיקה תהפוך לנפוצה, יישום עם ממשקים המשתנים באופן תמידי עלול להוביל לבעיות שימושיות, כאשר משתמשים עשויים להתקשות למצוא אלמנטים מוכרים או לנווט ביעילות.
- מציאת האיזון בין התאמה אישית לבין שמירה על ממשק עקבי וניתן ללמידה תהיה קריטית.

4. **הסתמכות יתר על בינה מלאכותית:**

- קיים סיכון של האצלת יתר של החלטות עיצוב למערכות בינה מלאכותית, מה שעלול להוביל לבחירות ממשק חסרות השראה, בעייתיות או פשוט שבורות.
- פיקוח אנושי והיכולת לעקוף עיצובים שנוצרו על ידי בינה מלאכותית יישארו חשובים בעתיד הנראה לעין.

5. **שיקולי נגישות:**

- הבטחת נגישותם של ממשקים המיוצרים באופן דינמי למשתמשים עם מוגבלויות מציבה אתגרים חדשים לחלוטין, מה שמדאיג במיוחד לאור רמת הציות הנמוכה לדרישות הנגישות במערכות טיפוסיות.

- מצד שני, ייתכן שמעצבי בינה מלאכותית יפותחו עם התחשבות מוכנית בנגישות, ויכולות ליצירת ממשקים נגישים באופן מיידי, בדיוק כפי שהם בונים ממשקי משתמש עבור משתמשים ללא מוגבלות.
- בכל מקרה, יש לתכנן מערכות GenUI עם הנחיות נגישות מוצקות ותהליכי בדיקה.

6. **אמון המשתמש ושקיפות:**

- משתמשים עשויים לחוש אי נוחות עם ממשקים שנראה כי "יודעים יותר מדי" עליהם או משתנים בדרכים שהם אינם מבינים.
- מתן שקיפות לגבי האופן והסיבה שבה ממשקים מותאמים אישית יהיה חשוב לבניית אמון המשתמש.

תחזית עתידית והזדמנויות

העתיד של ממשק משתמש יוצר (GenUI) טומן בחובו הבטחה עצומה למהפכה באופן שבו אנו מתקשרים עם מוצרים ושירותים דיגיטליים. ככל שהטכנולוגיה הזו ממשיכה להתפתח, אנו יכולים לצפות לשינוי מרעיד אדמה באופן שבו ממשקי משתמש מתוכננים, מיושמים ונחווים. אני חושב ש-GenUI הוא התופעה שסוף סוף תדחוף את התוכנה שלנו לתחום שנחשב כיום למדע בדיוני.

אחד ההיבטים המרגשים ביותר של GenUI הוא הפוטנציאל שלו לשפר נגישות בקנה מידה גדול שחורג מעבר להבטחה שאנשים עם מוגבלויות חמורות לא יהיו מודרים לחלוטין משימוש בתוכנה שלכם. על ידי התאמה אוטומטית של ממשקים לצרכים האישיים של כל משתמש, GenUI יכול להפוך חוויות דיגיטליות למכלילות יותר מאי פעם. דמיינו ממשקים שמתאימים את עצמם באופן חלק כדי לספק טקסט גדול יותר למשתמשים צעירים או לקויי ראייה, או פריסות פשוטות יותר עבור אנשים עם מוגבלויות קוגניטיביות, הכל ללא צורך בהגדרה ידנית או גרסאות "נגישות" נפרדות של יישומים.

יכולות ההתאמה האישית של GenUI צפויות להגביר את מעורבות המשתמשים, שביעות הרצון והנאמנות במגוון רחב של מוצרים דיגיטליים. ככל שהממשקים יהיו מכוונים יותר להעדפות ולהתנהגויות אישיות, משתמשים ימצאו את החוויות הדיגיטליות אינטואיטיביות ומהנות יותר, מה שעשוי להוביל לאינטראקציות עמוקות ומשמעותיות יותר עם הטכנולוגיה.

ל-GenUI יש גם את הפוטנציאל לשנות את תהליך החניכה עבור משתמשים חדשים. על ידי יצירת חוויות משתמש ראשוניות אינטואיטיביות ומותאמות אישית המסתגלות במהירות לרמת המומחיות של כל משתמש, GenUI יכול להפחית משמעותית את עקומת הלמידה הקשורה ליישומים חדשים. זה עשוי להוביל לקצב אימוץ מהיר יותר וביטחון מוגבר של המשתמשים בחקירת תכונות ופונקציונליות חדשות.

אפשרות מרגשת נוספת היא יכולתו של GenUI לשמור על חוויית משתמש עקבית במכשירים ופלטפורמות שונות, תוך כדי אופטימיזציה לכל הקשר שימוש ספציפי. זה עשוי לפתור את האתגר ארוך הטווח של מתן חוויות קוהרנטיות במרחב המכשירים המבוזר והגדל, החל מטלפונים חכמים וטאבלטים ועד למחשבים נייחים וטכנולוגיות מתפתחות כמו משקפי מציאות רבודה

האופי מונע-הנתונים של GenUI פותח הזדמנויות לאיטרציה ושיפור מהיר בעיצוב ממשק המשתמש. באמצעות איסוף נתונים בזמן אמת על אופן האינטראקציה של משתמשים עם ממשקים מחוללים, מעצבים ומפתחים יכולים לקבל תובנות חסרות תקדים על התנהגות והעדפות המשתמשים. לולאת משוב זו עשויה להוביל לשיפורים מתמשכים בעיצוב ממשק המשתמש, המונעים מדפוסי שימוש בפועל במקום מהנחות או בדיקות משתמש מוגבלות.

כדי להתכונן לשינוי זה, מעצבים יצטרכו לפתח את מערך הכישורים והתפיסות שלהם. המיקוד יעבור מיצירת פריסות קבועות לפיתוח מערכות עיצוב והנחיות מקיפות שיכולות להנחות את יצירת הממשק המונעת על ידי בינה מלאכותית. מעצבים יצטרכו לפתח הבנה עמוקה בניתוח נתונים, טכנולוגיות בינה מלאכותית וחשיבה מערכתית כדי להנחות ביעילות מערכות .GenUI

יתר על כן, כאשר GenUI מטשטש את הקווים בין עיצוב וטכנולוגיה, מעצבים יצטרכו לשתף פעולה באופן הדוק יותר עם מפתחים ומדעני נתונים. גישה בין-תחומית זו תהיה קריטית ביצירת מערכות GenUI שהן לא רק מושכות ויזואלית וידידותיות למשתמש, אלא גם חזקות מבחינה טכנית ואתית.

ההשלכות האתיות של GenUI יעלו גם הן לחזית ככל שהטכנולוגיה מתבגרת. למעצבים יהיה תפקיד מכריע בפיתוח מסגרות לשימוש אחראי בבינה מלאכותית בעיצוב ממשקים, תוך הבטחה שהפרסונליזציה משפרת את חוויות המשתמש מבלי לפגוע בפרטיות או לתמרן התנהגות משתמשים בדרכים לא אתיות.

במבט לעתיד, GenUI מציג הן הזדמנויות מרגשות והן אתגרים משמעותיים. יש לו פוטנציאל

ליצור חוויות דיגיטליות אינטואיטיביות, יעילות ומספקות יותר עבור משתמשים ברחבי העולם. בעוד שהוא ידרוש ממעצבים להסתגל ולרכוש כישורים חדשים, הוא גם מציע הזדמנות חסרת תקדים לעצב את עתיד האינטראקציה בין אדם למחשב בדרכים עמוקות ומשמעותיות. המסע לקראת מערכות GenUI מפותחות במלואן יהיה ללא ספק מורכב, אך הפוטנציאל בהיבט של שיפור חוויות משתמש ונגישות דיגיטלית הופך אותו לעתיד ששווה לשאוף אליו.

תזמור תהליכי עבודה חכם

בתחום פיתוח היישומים , תהליכי העבודה ממלאים תפקיד מכריע בהגדרת האופן שבו משימות, תהליכים ואינטראקציות משתמש מובנים ומבוצעים. ככל שהיישומים הופכים למורכבים יותר וציפיות המשתמשים ממשיכות לעלות, הצורך בתזמור תהליכי עבודה חכם ומסתגל הופך לברור יותר ויותר.

גישת "תזמור תהליכי העבודה החכם" מתמקדת בניצול רכיבי בינה מלאכותית כדי לתזמר ולייעל באופן דינמי תהליכי עבודה מורכבים בתוך יישומים. המטרה היא ליצור יישומים שהם יעילים יותר, מגיבים יותר ומסתגלים יותר לנתונים ולהקשר בזמן אמת.

בפרק זה, נחקור את העקרונות והדפוסים המרכזיים שעומדים בבסיס גישת תזמור תהליכי העבודה החכם. נבחן כיצד ניתן להשתמש בבינה מלאכותית כדי לנתב משימות באופן חכם,

לאוטמט קבלת החלטות ולהתאים תהליכי עבודה באופן דינמי בהתבסס על גורמים שונים כמו התנהגות משתמשים, ביצועי מערכת וכללים עסקיים . באמצעות דוגמאות מעשיות ותרחישים מהעולם האמיתי, נדגים את הפוטנציאל המשנה של בינה מלאכותית בייעול ואופטימיזציה של תהליכי עבודה ביישומים.

בין אם אתם בונים יישומים ארגוניים עם תהליכים עסקיים מורכבים או יישומים המיועדים לצרכנים עם מסעות משתמש דינמיים, הדפוסים והטכניקות שנדונים בפרק זה יציידו אתכם בידע ובכלים ליצירת תהליכי עבודה חכמים ויעילים המשפרים את חווית המשתמש הכוללת ומניעים ערך עסקי.

צורך עסקי

גישות מסורתיות לניהול תהליכי עבודה מסתמכות לעתים קרובות על כללים מוגדרים מראש ועצי החלטה סטטיים, שעלולים להיות נוקשים, בלתי גמישים ובלתי מסוגלים להתמודד עם האופי הדינמי של יישומים מודרניים.

שקלו תרחיש שבו יישום מסחר אלקטרוני צריך לטפל בתהליך מילוי הזמנות מורכב. תהליך העבודה עשוי לכלול מספר שלבים כמו אימות הזמנה, בדיקת מלאי, עיבוד תשלומים, משלוח והודעות ללקוח. לכל שלב עשויים להיות כללים משלו, תלויות, אינטגרציות חיצוניות ומנגנוני טיפול בחריגים. ניהול תהליך עבודה כזה באופן ידני או באמצעות לוגיקה מקודדת מראש יכול במהירות להפוך למסורבל, מועד לטעויות וקשה לתחזוקה.

יתר על כן, ככל שהיישום מתרחב ומספר המשתמשים הבו-זמניים גדל, תהליך העבודה עשוי להזדקק להתאמה ואופטימיזציה של עצמו בהתבסס על נתונים בזמן אמת וביצועי המערכת. לדוגמה, בתקופות של עומס שיא, היישום עשוי להזדקק להתאמה דינמית של תהליך העבודה כדי לתעדף משימות מסוימות, להקצות משאבים ביעילות ולהבטיח חוויית משתמש חלקה.

כאן נכנסת לתמונה גישת "תזמור תהליכי העבודה החכם". באמצעות שימוש ברכיבי בינה מלאכותית, מפתחים יכולים ליצור תהליכי עבודה חכמים, מסתגלים ובעלי יכולת אופטימיזציה עצמית. בינה מלאכותית יכולה לנתח כמויות עצומות של נתונים, ללמוד מניסיון העבר ולקבל החלטות מושכלות בזמן אמת כדי לתזמר את תהליך העבודה ביעילות.

יתרונות מרכזיים

1. **יעילות מוגברת:** בינה מלאכותית יכולה לייעל הקצאת משימות, ניצול משאבים וביצוע תהליכי עבודה, מה שמוביל לזמני עיבוד מהירים יותר ושיפור היעילות הכוללת.
2. **יכולת הסתגלות:** תהליכי עבודה מונעי בינה מלאכותית יכולים להסתגל באופן דינמי לתנאים משתנים, כגון תנודות בביקוש המשתמשים, ביצועי המערכת או דרישות עסקיות, תוך הבטחת תגובתיות ועמידות היישום.
3. **קבלת החלטות אוטומטית:** בינה מלאכותית יכולה לאוטומט תהליכי קבלת החלטות מורכבים בתוך תהליך העבודה, להפחית התערבות ידנית ולמזער את הסיכון לטעויות אנוש.
4. **התאמה אישית:** בינה מלאכותית יכולה לנתח התנהגות משתמשים, העדפות והקשר כדי להתאים אישית את תהליך העבודה ולספק חוויות מותאמות למשתמשים בודדים.
5. **סקיילויות:** תהליכי עבודה מבוססי בינה מלאכותית יכולים להתרחב באופן חלק כדי לטפל בנפחים גדלים של נתונים ואינטראקציות משתמשים, מבלי לפגוע בביצועים או באמינות.

בסעיפים הבאים, נחקור את התבניות והטכניקות המרכזיות המאפשרות את היישום של תהליכי עבודה חכמים ונציג דוגמאות מהעולם האמיתי לאופן שבו בינה מלאכותית משנה את ניהול תהליכי העבודה ביישומים מודרניים.

תבניות מרכזיות

כדי ליישם תזמור תהליכי עבודה חכמים ביישומים, מפתחים יכולים לנצל מספר תבניות מרכזיות המרתמות את כוחה של הבינה המלאכותית. תבניות אלו מספקות גישה מובנית לתכנון וניהול תהליכי עבודה, ומאפשרות ליישומים להסתגל, לייעל ולאוטומט תהליכים בהתבסס על נתונים והקשר בזמן אמת. הבה נחקור כמה מהתבניות היסודיות בתזמור תהליכי עבודה חכמים.

ניתוב משימות דינמי

תבנית זו כוללת שימוש בבינה מלאכותית לניתוב חכם של משימות בתוך תהליך העבודה בהתבסס על גורמים שונים כגון עדיפות המשימה, זמינות משאבים וביצועי המערכת. אלגוריתמים של בינה מלאכותית יכולים לנתח את המאפיינים של כל משימה, לשקול את המצב הנוכחי של המערכת ולקבל החלטות מושכלות להקצאת משימות למשאבים או נתיבי עיבוד המתאימים ביותר. ניתוב משימות דינמי מבטיח שמשימות מופצות ומבוצעות ביעילות, תוך אופטימיזציה של ביצועי תהליך העבודה הכולל.

```
class TaskRouter
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  attr_accessor :task

  its at entirely AI the by called be can that functions of list #
  received task the on depending discretion #

  function :analyze_task_priority do
    TaskPriorityAnalyzer.perform(task)
  end

  function :check_resource_availability, ... #
  function :assess_system_performance, ... #
  function :assign_task_to_resource, ... #

  DIRECTIVE = "intelligently for responsible ,router task a are You
resource ,priority on based resources available to tasks assigning
performance... system and ,availability   "

  def initialize(task)
    self.task = task
    transcript << { system: DIRECTIVE }
    transcript << { user: task.to_json }
  end

  def perform
    while task.unassigned?
      chat_completion
```

```
31
32        break and counter loop max add todo: #
33      end
34
35      analysis later for transcript the capture #
36      task.update(routing_transcript: transcript)
37    end
38  end
```

שימו לב ללולאה שנוצרת על ידי ביטוי ה-while בשורה 29, אשר ממשיכה לבקש מהבינה המלאכותית עד להקצאת המשימה. בשורה 35, אנו שומרים את התמליל של המשימה לצורך ניתוח עתידי, אם יהיה בכך צורך.

קבלת החלטות מבוססת הקשר

באפשרותכם להשתמש בקוד דומה מאוד כדי לקבל החלטות מודעות הקשר בתוך תהליך העבודה. על ידי ניתוח נקודות מידע רלוונטיות כגון העדפות משתמש, דפוסים היסטוריים , וקלט בזמן אמת, רכיבי הבינה המלאכותית יכולים לקבוע את מסלול הפעולה המתאים ביותר בכל נקודת החלטה בתהליך העבודה. התאימו את התנהגות תהליך העבודה שלכם בהתבסס על ההקשר הספציפי של כל משתמש או תרחיש, תוך מתן חוויות מותאמות אישית ומיטביות.

הרכבת תהליך עבודה מסתגל

דפוס זה מתמקד בהרכבה והתאמה דינמית של תהליכי עבודה בהתבסס על דרישות או תנאים משתנים. בינה מלאכותית יכולה לנתח את המצב הנוכחי של תהליך העבודה, לזהות צווארי בקבוק או חוסר יעילות, ולשנות באופן אוטומטי את מבנה תהליך העבודה כדי למטב את הביצועים. הרכבת תהליך עבודה מסתגל מאפשרת ליישומים להתפתח ולשפר את תהליכיהם באופן מתמיד ללא צורך בהתערבות ידנית.

טיפול בחריגים והתאוששות

טיפול בחריגים והתאוששות הם היבטים קריטיים בתזמור תהליכי עבודה חכמים. בעבודה עם רכיבי בינה מלאכותית ותהליכי עבודה מורכבים, חיוני לצפות ולטפל בחריגים בצורה

חלקה כדי להבטיח את היציבות והאמינות של המערכת.

להלן כמה שיקולים וטכניקות מרכזיים לטיפול בחריגים והתאוששות בתהליכי עבודה חכמים:

1. **הפצת חריגים:** יישמו גישה עקבית להפצת חריגים בין רכיבי תהליך העבודה. כאשר מתרחש חריג בתוך רכיב, יש לתפוס אותו, לתעד אותו, ולהפיץ אותו למתזמר או לרכיב נפרד האחראי על טיפול בחריגים . הרעיון הוא למרכז את הטיפול בחריגים ולמנוע מחריגים להיבלע בשקט, וכן לפתוח אפשרויות לטיפול חכם בשגיאות.
2. **מנגנוני ניסיון חוזר:** מנגנוני ניסיון חוזר עוזרים לשפר את החוסן של תהליך העבודה ולטפל בכשלים זמניים בצורה חלקה. בהחלט כדאי ליישם מנגנוני ניסיון חוזר עבור חריגים זמניים או הניתנים לשחזור, כגון בעיות קישוריות רשת או אי-זמינות משאבים שניתן לנסות אותם מחדש באופן אוטומטי לאחר השהיה מוגדרת. כאשר יש מתזמר או מטפל חריגים מבוסס בינה מלאכותית, אסטרטגיות הניסיון החוזר שלכם אינן חייבות להיות מכניות באופיין, ולהסתמך על אלגוריתמים קבועים כמו נסיגה מעריכית. אתם יכולים להשאיר את הטיפול בניסיון החוזר ל"שיקול דעתו" של רכיב הבינה המלאכותית האחראי להחליט כיצד לטפל בחריג.
3. **אסטרטגיות גיבוי:** אם רכיב AI נכשל במתן תגובה תקפה או נתקל בשגיאה—תופעה נפוצה בהתחשב בטבעו החדשני—יש להכין מנגנון גיבוי כדי להבטיח שתזרים העבודה יכול להמשיך. זה יכול לכלול שימוש בערכי ברירת מחדל, אלגוריתמים חלופיים, או אדם במעגל התהליך לקבלת החלטות והמשך התקדמות תזרים העבודה.
4. **פעולות פיצוי:** הנחיות המתזמר צריכות לכלול הוראות לגבי פעולות פיצוי לטיפול בחריגים שלא ניתן לפתור באופן אוטומטי. פעולות פיצוי הן צעדים שננקטים כדי לבטל או למתן את ההשפעות של פעולה שנכשלה. לדוגמה, אם שלב עיבוד תשלום נכשל, פעולת פיצוי יכולה להיות ביטול העסקה ויידוע המשתמש. פעולות פיצוי עוזרות לשמור על עקביות ושלמות הנתונים בעת התמודדות עם חריגים.
5. **ניטור והתראת חריגים:** הגדר מנגנוני ניטור והתראה לזיהוי ויידוע בעלי העניין הרלוונטיים על חריגים קריטיים. ניתן להגדיר למתזמר ספי התראה וכללים להפעלת התראות כאשר חריגים חורגים ממגבלות מסוימות או כאשר מתרחשים סוגים ספציפיים של חריגים. זה מאפשר זיהוי ופתרון פרואקטיבי של בעיות לפני שהן משפיעות על המערכת הכוללת.

הנה דוגמה לטיפול והתאוששות מחריגים ברכיב תזרים עבודה ב-Ruby:

```ruby
class InventoryManager
  def check_availability(order)
    begin
      \logic check inventory Perform #
      inventory = Inventory.find_by(product_id: order.product_id)
      if inventory.available_quantity >= order.quantity
        return true
      else
        raise InsufficientInventoryError,
            " product for inventory Insufficient#{order.product_id}"
      end
    rescue InsufficientInventoryError => e
      exception the Log #
      logger.error(" failed: check Inventory#{e.message}")

      delay a after operation the Retry #
      retry_count ||= 0
      if retry_count < MAX_RETRIES
        retry_count += 1
        sleep(RETRY_DELAY)
        retry
      else
        intervention manual to Fallback #
        NotificationService.admin(" Order failed: check Inventory#{order.id}")
        return false
      end
    end
  end
end
```

בדוגמה זו, הרכיב InventoryManager בודק את זמינות המוצר עבור הזמנה נתונה. אם הכמות הזמינה אינה מספקת, הוא מעלה שגיאת InsufficientInventoryError. החריגה נתפסת, נרשמת ביומן, ומיושם מנגנון ניסיון חוזר. אם חורגים ממגבלת הניסיונות החוזרים, הרכיב עובר להתערבות ידנית על ידי שליחת התראה למנהל המערכת.

על ידי יישום מנגנוני טיפול והתאוששות מחריגים חזקים, ניתן להבטיח שתהליכי העבודה החכמים שלך יהיו עמידים, ברי-תחזוקה, ומסוגלים להתמודד עם מצבים בלתי צפויים בצורה

אלגנטית.

דפוסים אלה מהווים את הבסיס לתזמור תהליכי עבודה חכמים וניתן לשלב ולהתאים אותם כדי לענות על הדרישות הספציפיות של יישומים שונים. באמצעות שימוש בדפוסים אלה, מפתחים יכולים ליצור תהליכי עבודה גמישים, עמידים וממוטבים לביצועים וחוויית משתמש.

בחלק הבא, נחקור כיצד ניתן ליישם דפוסים אלה בפועל, תוך שימוש בדוגמאות מהעולם האמיתי וקטעי קוד להמחשת השילוב של רכיבי בינה מלאכותית בניהול תהליכי עבודה.

יישום תזמור תהליכי עבודה חכמים בפועל

עכשיו שחקרנו את הדפוסים המרכזיים בתזמור תהליכי עבודה חכמים, בואו נצלול לאופן שבו ניתן ליישם דפוסים אלה ביישומים מהעולם האמיתי. נספק דוגמאות מעשיות וקטעי קוד להמחשת השילוב של רכיבי בינה מלאכותית בניהול תהליכי עבודה.

מעבד הזמנות חכם

בואו נצלול לדוגמה מעשית של יישום תזמור תהליכי עבודה חכמים באמצעות רכיב OrderProcessor מבוסס בינה מלאכותית ביישום מסחר אלקטרוני של Rails on Ruby. ה-OrderProcessor מממש את רעיון מנהל תהליכים לאינטגרציה ארגונית שפגשנו לראשונה בפרק 3 כשדנו בריבוי עובדים. הרכיב יהיה אחראי על ניהול תהליך מימוש ההזמנות, קבלת החלטות ניתוב על בסיס תוצאות ביניים, ותזמור הביצוע של שלבי עיבוד שונים.

תהליך מימוש ההזמנה כולל מספר שלבים כגון אימות הזמנה, בדיקת מלאי, עיבוד תשלום ומשלוח. כל שלב מיושם כתהליך עובד נפרד שמבצע משימה ספציפית ומחזיר את התוצאה ל-OrderProcessor. השלבים אינם חובה, ואפילו לא בהכרח צריכים להתבצע בסדר מדויק.

הנה דוגמת יישום של ה-OrderProcessor. הוא כולל שני מיקסינים מ-Raix. הראשון (ChatCompletion) מעניק לו את היכולת לבצע השלמת צ'אט, מה שהופך אותו לרכיב בינה מלאכותית. השני (FunctionDispatch) מאפשר קריאה לפונקציות על ידי הבינה המלאכותית, מה שמאפשר לה להגיב לפרומפט עם הפעלת פונקציה במקום הודעת טקסט.

פונקציות העבודה (order_validate, inventory_check, וכדומה) מאצילות את הפעולות למחלקות העבודה המתאימות, אשר יכולות להיות רכיבי בינה מלאכותית או רכיבים שאינם בינה מלאכותית, כאשר הדרישה היחידה היא שהן יחזירו את תוצאות עבודתן בפורמט שניתן להציג כמחרוזת.

בדומה לכל הדוגמאות האחרות בחלק זה של הספר, קוד זה הוא למעשה פסאודו-קוד ומטרתו רק להעביר את משמעות התבנית ולעורר השראה ליצירות שלכם. תיאורים מלאים של תבניות ודוגמאות קוד שלמות נכללים בחלק 2.

```
class OrderProcessor
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "with... tasked ,processor order an are You"

  def initialize(order)
    self.order = order
    transcript << { system: SYSTEM_DIRECTIVE }
    transcript << { user: order.to_json }
  end

  def perform
    called is ‘‘stop_looping! until looping continue will #
    chat_completion(loop: true)
  end

  AI the by called be to available functions of list #
  brevity for truncated #

  def functions
    [
      {
        name: "validate_order",
        description: "order of validity check to Invoke",
        parameters: {
          ...
      },
      ...
```

```
    ]
  end

  AI the by called be can that functions of implementation #
  order the of needs the on depending ,discretion its at entirely #

  def validate_order
    OrderValidationWorker.perform(@order)
  end

  def check_inventory
    InventoryCheckWorker.perform(@order)
  end

  def process_payment
    PaymentProcessingWorker.perform(@order)
  end

  def schedule_shipping
    ShippingSchedulerWorker.perform(@order)
  end

  def send_confirmation
    OrderConfirmationWorker.perform(@order)
  end

  def finished_processing
    @order.update!(transcript:, processed_at: Time.current)
    stop_looping!
  end
end
```

בדוגמה, ה-OrderProcessor מאותחל עם אובייקט הזמנה ושומר תמליל של ביצוע תהליך העבודה, בפורמט תמליל שיחה טיפוסי שהוא טבעי למודלים של שפה גדולה. ניתנת שליטה מלאה ל-AI לתזמר את הביצוע של שלבי העיבוד השונים, כגון אימות הזמנה, בדיקת מלאי, עיבוד תשלום ומשלוח.

בכל פעם שהמתודה completion_chat נקראת, התמליל נשלח ל-AI כדי שיספק השלמה כקריאת פונקציה. זה תלוי לחלוטין ב-AI לנתח את התוצאה של השלב הקודם ולקבוע את הפעולה המתאימה לביצוע. לדוגמה, אם בדיקת המלאי מגלה רמות מלאי נמוכות, ה-

OrderProcessor יכול לתזמן משימת מילוי מחדש. אם עיבוד התשלום נכשל, הוא יכול ליזום ניסיון חוזר או להודיע לתמיכת לקוחות.

> הדוגמה לעיל אינה כוללת פונקציות מוגדרות למילוי מחדש או להודעה לתמיכת לקוחות, אבל בהחלט יכולה לכלול אותן.

התמליל גדל בכל פעם שפונקציה נקראת ומשמש כרישום של ביצוע תהליך העבודה, כולל התוצאות של כל שלב וההוראות שנוצרו על ידי ה-AI לשלבים הבאים. ניתן להשתמש בתמליל זה לצורך ניפוי באגים, ביקורת ומתן שקיפות לתהליך מילוי ההזמנות.

על ידי ניצול ה-AI ב-OrderProcessor, אפליקציית המסחר האלקטרוני יכולה להתאים את תהליך העבודה באופן דינמי בהתבסס על נתונים בזמן אמת ולטפל בחריגים באופן חכם. רכיב ה-AI יכול לקבל החלטות מושכלות, לייעל את תהליך העבודה ולהבטיח עיבוד הזמנות חלק גם בתרחישים מורכבים.

העובדה שהדרישה היחידה מתהליכי העבודה היא להחזיר פלט מובן כלשהו שה-AI יוכל לשקול בעת החלטה מה לעשות הלאה, עשויה להתחיל להבהיר לכם כיצד גישה זו יכולה להפחית את עבודת מיפוי הקלט/פלט שבדרך כלל מעורבת בשילוב מערכות נפרדות זו עם זו.

מנטר תוכן חכם

אפליקציות מדיה חברתית בדרך כלל דורשות לפחות ניטור תוכן מינימלי כדי להבטיח קהילה בטוחה ובריאה. דוגמת רכיב ה-ContentModerator הזה מנצל AI כדי לתזמר באופן חכם את תהליך הניטור, מקבל החלטות בהתבסס על מאפייני התוכן והתוצאות של שלבי ניטור שונים.

תהליך הניטור כולל מספר שלבים כמו ניתוח טקסט, זיהוי תמונות, הערכת מוניטין משתמש וסקירה ידנית. כל שלב מיושם כתהליך עבודה נפרד שמבצע משימה ספציפית ומחזיר את התוצאה ל-ContentModerator.

הנה דוגמת יישום של ה-ContentModerator:

```
class ContentModerator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = ",manager process moderator content a are You
content… user-generated moderating in involved workflow the with tasked  "

  def initialize(content)
    @content = content
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: content.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  AI the by called be to available functions of list #
  brevity for truncated #

  def functions
    [
      {
        name: "analyze_text",
        ... #
      },
      {
        name: "recognize_image",
        description: "images… describe to Invoke",
        ... #
      },
      {
        name: "assess_user_reputation",
        ... #
      },
      {
```

```
      name: "escalate_to_manual_review",
      ... #
    },
    {
      name: "approve_content",
      ... #
    },
    {
      name: "reject_content",
      ... #
    }
  ]
end

AI the by called be can that functions of implementation #
order the of needs the on depending ,discretion its at entirely #

def analyze_text
  result = TextAnalysisWorker.perform(@content)
  continue_with(result)
end

def recognize_image
  result = ImageRecognitionWorker.perform(@content)
  continue_with(result)
end

def assess_user_reputation
  result = UserReputationWorker.perform(@content.user)
  continue_with(result)
end

def escalate_to_manual_review
  ManualReviewWorker.perform(@content)
  @content.update!(status: 'pending', transcript: @transcript)
end

def approve_content
  @content.update!(status: 'approved', transcript: @transcript)
end

def reject_content
```

```
85      @content.update!(status: 'rejected', transcript: @transcript)
86    end
87
88    private
89
90    def continue_with(result)
91      @transcript << { function: result }
92      complete(@transcript)
93    end
94  end
```

בדוגמה זו, ה-ContentModerator מאותחל עם אובייקט תוכן ומנהל תמליל ניהול בפורמט של שיחה. רכיב הבינה המלאכותית שולט באופן מלא בתהליך הניהול, ומחליט אילו שלבים לבצע בהתבסס על מאפייני התוכן והתוצאות של כל שלב.

פונקציות העבודה הזמינות שהבינה המלאכותית יכולה להפעיל כוללות את text_analyze, image_recognize, reputation_user_assess ו-review_manual_to_escalate. כל פונקציה מעבירה את המשימה לתהליך העבודה המתאים (TextAnalysisWorker, ImageRecognitionWorker, וכו') ומוסיפה את התוצאה לתמליל הניהול, למעט פונקציית ההסלמה, שמשמשת כמצב סופי. לבסוף, הפונקציות content_approve ו-content_reject גם הן משמשות כמצבים סופיים.

רכיב הבינה המלאכותית מנתח את התוכן ומחליט על הפעולה המתאימה לביצוע. אם התוכן מכיל הפניות לתמונות, הוא יכול לקרוא לעובד image_recognize לסיוע בסקירה חזותית. אם עובד כלשהו מתריע על תוכן שעלול להיות מזיק, הבינה המלאכותית עשויה להחליט להעביר את התוכן לבדיקה ידנית או פשוט לדחות אותו לחלוטין. אך בהתאם לחומרת האזהרה, הבינה המלאכותית עשויה לבחור להשתמש בתוצאות הערכת המוניטין של המשתמש בהחלטה כיצד לטפל בתוכן שאינה בטוחה לגביו. בהתאם למקרה השימוש, ייתכן שלמשתמשים אמינים יש יותר גמישות במה שהם יכולים לפרסם. וכן הלאה...

בדומה לדוגמת מנהל התהליכים הקודמת, תמליל הניהול משמש כרישום של ביצוע תהליך העבודה, כולל תוצאות כל שלב וההחלטות שנוצרו על ידי הבינה המלאכותית. ניתן להשתמש בתמליל זה לביקורת, שקיפות ושיפור תהליך הניהול לאורך זמן.

באמצעות שימוש בבינה מלאכותית ב-ContentModerator, אפליקציית המדיה החברתית

יכולה להתאים באופן דינמי את תהליך הניהול בהתבסס על מאפייני התוכן ולטפל בתרחישי ניהול מורכבים באופן חכם. רכיב הבינה המלאכותית יכול לקבל החלטות מושכלות, לייעל את תהליך העבודה ולהבטיח חוויית קהילה בטוחה ובריאה.

הבה נחקור שתי דוגמאות נוספות המדגימות תזמון משימות חזוי וטיפול בחריגים והתאוששות במסגרת תזמור תהליכי עבודה חכם.

תזמון משימות חזוי במערכת תמיכת לקוחות

באפליקציית תמיכת לקוחות שנבנתה עם , ניהול ותעדוף יעיל של פניות תמיכה הוא קריטי למתן סיוע מהיר ללקוחות. רכיב ה-SupportTicketScheduler מנצל בינה מלאכותית כדי לתזמן ולהקצות באופן חזוי פניות תמיכה לנציגים זמינים בהתבסס על גורמים שונים כמו דחיפות הפנייה, מומחיות הנציג ועומס העבודה.

```
class SupportTicketScheduler
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = ",scheduler ticket support a are You
agents... available to tickets assigning intelligently with tasked    "

  def initialize(ticket)
    @ticket = ticket
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: ticket.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
```

```
    [
      {
        name: "analyze_ticket_urgency",
        ... #
      },
      {
        name: "list_available_agents",
        description: "agents available of expertise Includes",
        ... #
      },
      {
        name: "predict_agent_workload",
        description: "workloads upcoming predict to data historical Uses",
        ... #
      },
      {
        name: "assign_ticket_to_agent",
        ... #
      },
      {
        name: "reschedule_ticket",
        ... #
      }
    ]
  end

  AI the by called be can that functions of implementation #
  order the of needs the on depending ,discretion its at entirely #

  def analyze_ticket_urgency
    result = TicketUrgencyAnalyzer.perform(@ticket)
    continue_with(result)
  end

  def list_available_agents
    result = ListAvailableAgents.perform
    continue_with(result)
  end

  def predict_agent_workload
    result = AgentWorkloadPredictor.perform
    continue_with(result)
```

```
    end

    def assign_ticket_to_agent
      TicketAssigner.perform(@ticket, @transcript)
    end

    def delay_assignment(until)
      until = DateTimeStandardizer.process(until)
      SupportTicketScheduler.delay(@ticket, @transcript, until)
    end

    private

    def continue_with(result)
      @transcript << { function: result }
      complete(@transcript)
    end
  end
```

בדוגמה זו, ה-SupportTicketScheduler מאותחל עם אובייקט כרטיס תמיכה ומנהל תמליל תזמון. רכיב הבינה המלאכותית מנתח את פרטי הכרטיס ומתזמן באופן חזוי את שיוך הכרטיס בהתבסס על גורמים כמו דחיפות הכרטיס, מומחיות הנציג ועומס העבודה החזוי של הנציג.

הפונקציות הזמינות שהבינה המלאכותית יכולה להפעיל כוללות את urgency_ticket_analyze, agents_available_list, workload_agent_predict, ו-agent_to_ticket_assign. כל פונקציה מעבירה את המשימה לרכיב מנתח או חוזה מתאים ומוסיפה את התוצאה לתמליל התזמון. לבינה המלאכותית יש גם אפשרות לעכב את השיוך באמצעות הפונקציה assignment_delay.

רכיב הבינה המלאכותית בוחן את תמליל התזמון ומקבל החלטות מושכלות לגבי שיוך הכרטיס. הוא מתחשב בדחיפות הכרטיס, במומחיות של הנציגים הזמינים ובעומס העבודה החזוי של כל נציג כדי לקבוע את הנציג המתאים ביותר לטיפול בכרטיס.

באמצעות ניצול תזמון משימות חזוי, יישום התמיכה בלקוחות יכול למטב את שיוך הכרטיסים, להפחית זמני תגובה ולשפר את שביעות רצון הלקוחות הכללית. ניהול יעיל ופרואקטיבי של כרטיסי תמיכה מבטיח ששיוך הכרטיסים הנכונים לנציגים הנכונים נעשה בזמן הנכון.

טיפול בחריגים והתאוששות בצינור עיבוד נתונים

טיפול בחריגים והתאוששות מכשלים הם חיוניים להבטחת שלמות הנתונים ומניעת אובדן נתונים. רכיב ה-DataProcessingOrchestrator משתמש בבינה מלאכותית כדי לטפל בחריגים באופן חכם ולתזמר את תהליך ההתאוששות בצינור עיבוד נתונים

```
class DataProcessingOrchestrator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "orchestrator... processing data a are You"

  def initialize(data_batch)
    @data_batch = data_batch
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: data_batch.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "validate_data",
        ... #
      },
      {
        name: "process_data",
        ... #
      },
      {
        name: "request_fix",
```

```
      ... #
    },
    {
      name: "retry_processing",
      ... #
    },
    {
      name: "mark_data_as_failed",
      ... #
    },
    {
      name: "finished",
      ... #
    }
  ]
end

AI the by called be can that functions of implementation #
order the of needs the on depending ,discretion its at entirely #

def validate_data
  result = DataValidator.perform(@data_batch)
  continue_with(result)
rescue ValidationException => e
  handle_validation_exception(e)
end

def process_data
  result = DataProcessor.perform(@data_batch)
  continue_with(result)
rescue ProcessingException => e
  handle_processing_exception(e)
end

def request_fix(description_of_fix)
  result = SmartDataFixer.new(description_of_fix, @data_batch)
  continue_with(result)
end

def retry_processing(timeout_in_seconds)
  wait(timeout_in_seconds)
  process_data
```

```
77    end
78
79    def mark_data_as_failed
80      @data_batch.update!(status: 'failed', transcript: @transcript)
81    end
82
83    def finished
84      @data_batch.update!(status: 'finished', transcript: @transcript)
85    end
86
87    private
88
89    def continue_with(result)
90      @transcript << { function: result }
91      complete(@transcript)
92    end
93
94    def handle_validation_exception(exception)
95      @transcript << { exception: exception.message }
96      complete(@transcript)
97    end
98
99    def handle_processing_exception(exception)
100     @transcript << { exception: exception.message }
101     complete(@transcript)
102   end
103 end
```

בדוגמה זו, ה-DataProcessingOrchestrator מאותחל עם אובייקט אצוות נתונים ומנהל תמליל עיבוד. רכיב הבינה המלאכותית מתזמר את צינור עיבוד הנתונים, מטפל בחריגים ומתאושש מכשלים לפי הצורך.

הפונקציות הזמינות עבור הבינה המלאכותית להפעלה כוללות את data_validate, process_data, fix_request, processing_retry ו-failed_as_data_mark. כל פונקציה מאצילה את המשימה לרכיב עיבוד נתונים מתאים ומוסיפה את התוצאה או פרטי החריגה לתמליל העיבוד.

אם מתרחשת חריגת אימות במהלך שלב data_validate, הפונקציה handle_validation_exception מוסיפה את נתוני החריגה לתמליל ומעבירה את השליטה בחזרה לבינה המלאכותית. באופן דומה, אם מתרחשת חריגת עיבוד במהלך שלב data_process, הבינה

המלאכותית יכולה להחליט על אסטרטגיית ההתאוששות.

בהתאם לאופי החריגה שנתקלה בה, הבינה המלאכותית יכולה לפי שיקול דעתה להחליט לקרוא ל-fix_request, אשר מאציל לרכיב SmartDataFixer המופעל על ידי בינה מלאכותית (ראה פרק נתונים בעלי יכולת תיקון עצמי). מתקן הנתונים מקבל תיאור בשפה פשוטה של האופן שבו עליו לשנות את batch_@data כדי שניתן יהיה לנסות את העיבוד מחדש. אולי ניסיון חוזר מוצלח יכלול הסרת רשומות מאצוות הנתונים שנכשלו באימות ו/או העתקתן לצינור עיבוד שונה לבדיקה אנושית? האפשרויות הן כמעט אינסופיות.

על ידי שילוב טיפול והתאוששות מחריגים מבוססי בינה מלאכותית, יישום עיבוד הנתונים הופך לחסין יותר ועמיד בפני תקלות. ה-DataProcessingOrchestrator מנהל חריגים באופן חכם, ממזער אובדן נתונים ומבטיח ביצוע חלק של תהליך עיבוד הנתונים.

ניטור ותיעוד

ניטור ותיעוד מספקים נראות להתקדמות, ביצועים ובריאות של רכיבי זרימת העבודה מבוססי הבינה המלאכותית, ומאפשרים למפתחים לעקוב ולנתח את התנהגות המערכת. יישום מנגנוני ניטור ותיעוד יעילים הוא חיוני לניפוי באגים, ביקורת ושיפור מתמשך של תהליכי עבודה חכמים.

ניטור התקדמות וביצועים של זרימת העבודה

כדי להבטיח ביצוע חלק של תהליכי עבודה חכמים, חשוב לנטר את ההתקדמות והביצועים של כל רכיב בזרימת העבודה. זה כולל מעקב אחר מדדים ואירועים מרכזיים לאורך מחזור החיים של זרימת העבודה.

היבטים חשובים לניטור כוללים:

1. **זמן ביצוע זרימת העבודה:** מדידת הזמן שלוקח לכל רכיב בזרימת העבודה להשלים את משימתו. זה עוזר לזהות צווארי בקבוק בביצועים ולייעל את יעילות זרימת העבודה הכוללת.

2. **ניצול משאבים:** ניטור השימוש במשאבי מערכת, כגון CPU, זיכרון ואחסון, על ידי כל רכיב בזרימת העבודה. זה עוזר להבטיח שהמערכת פועלת במסגרת הקיבולת שלה ויכולה להתמודד עם העומס ביעילות.

3. **שיעורי שגיאות וחריגות:** עקוב אחר התרחשות של שגיאות וחריגות בתוך רכיבי תזרים העבודה. זה מסייע בזיהוי בעיות פוטנציאליות ומאפשר טיפול מקדים בשגיאות והתאוששות.

4. **נקודות החלטה ותוצאות:** נטר את נקודות ההחלטה בתוך תזרים העבודה ואת התוצאות של החלטות מבוססות בינה מלאכותית. זה מספק תובנות לגבי ההתנהגות והיעילות של רכיבי הבינה המלאכותית.

הנתונים שנאספים על ידי תהליכי הניטור יכולים להיות מוצגים בלוחות מחוונים או לשמש כקלט לדוחות מתוזמנים המיידעים את מנהלי המערכת על מצב המערכת.

ניתן להזין נתוני ניטור לתהליך מנהל מערכת מבוסס בינה מלאכותית לצורך סקירה ופעולה פוטנציאלית!

תיעוד אירועים והחלטות מרכזיים

תיעוד הוא נוהל חיוני הכולל לכידה ואחסון של מידע רלוונטי אודות אירועים מרכזיים, החלטות וחריגות המתרחשים במהלך הרצת תזרים העבודה.

היבטים חשובים לתיעוד כוללים:

1. **התחלה וסיום של תזרים העבודה:** תעד את זמני ההתחלה והסיום של כל הרצת תזרים עבודה, יחד עם כל מטא-דאטה רלוונטי כגון נתוני הקלט והקשר המשתמש.

2. **הרצת רכיבים:** תעד את פרטי ההרצה של כל רכיב בתזרים העבודה, כולל פרמטרי הקלט, תוצאות הפלט, וכל נתוני ביניים שנוצרו.

3. **החלטות ושיקולי בינה מלאכותית:** תעד את ההחלטות שהתקבלו על ידי רכיבי הבינה המלאכותית, יחד עם השיקולים שמאחוריהן או ציוני הביטחון. זה מספק שקיפות ומאפשר ביקורת של החלטות מבוססות בינה מלאכותית.

4. **חריגות והודעות שגיאה:** תעד כל חריגה או הודעת שגיאה שנתקלו בהן במהלך הרצת תזרים העבודה, כולל מעקב המחסנית ומידע הקשר רלוונטי.

ניתן ליישם תיעוד באמצעות טכניקות שונות, כגון כתיבה לקבצי לוג, אחסון לוגים במסד נתונים, או שליחת לוגים לשירות תיעוד מרכזי. חשוב לבחור מסגרת תיעוד המספקת גמישות, יכולת הרחבה ואינטגרציה קלה עם ארכיטקטורת האפליקציה.

הנה דוגמה כיצד ניתן ליישם תיעוד באפליקציית Rails on Ruby באמצעות המחלקה :ActiveSupport::Logger

```
class WorkflowLogger
  def self.log(message, severity = :info)
    @logger ||= ActiveSupport::Logger.new('workflow.log')
    @logger.formatter ||= proc do |severity, datetime, progname, msg|
      "#{datetime}[ #{severity} ]#{msg}\n"
    end
    @logger.send(severity, message)
  end
end

example Usage #
WorkflowLogger.log(" order for initiated Workflow##{@order.id}")
WorkflowLogger.log("successfully completed processing Payment")
WorkflowLogger.log(" item for failed check Inventory##{item.id}", :error)
```

באמצעות מיקום אסטרטגי של הצהרות רישום לאורך רכיבי תהליך העבודה ונקודות ההחלטה של הבינה המלאכותית, מפתחים יכולים לאסוף מידע חיוני לצורך ניפוי באגים, ביקורת וניתוח.

יתרונות הניטור והרישום

יישום ניטור ורישום בתזמור תהליכי עבודה חכמים מציע מספר יתרונות:

1. **ניפוי באגים ופתרון בעיות:** יומנים מפורטים ונתוני ניטור מסייעים למפתחים לזהות ולאבחן בעיות במהירות. הם מספקים תובנות לגבי זרימת ביצוע תהליך העבודה, אינטראקציות בין רכיבים, וכל שגיאה או חריגה שנתקלו בה.

2. **אופטימיזציית ביצועים:** ניטור מדדי ביצועים מאפשר למפתחים לזהות צווארי בקבוק ולייעל את רכיבי תהליך העבודה לשיפור היעילות. באמצעות ניתוח זמני ביצוע, ניצול משאבים ומדדים אחרים, מפתחים יכולים לקבל החלטות מושכלות לשיפור הביצועים הכוללים של המערכת.

3. **ביקורת וציות:** רישום אירועים והחלטות מרכזיות מספק מסלול ביקורת לציות רגולטורי ואחריותיות. זה מאפשר לארגונים לעקוב ולאמת את הפעולות שננקטו על ידי רכיבי הבינה המלאכותית ולהבטיח עמידה בכללים העסקיים ובדרישות החוק.

4. **שיפור מתמיד:** נתוני ניטור ורישום משמשים כקלט חשוב לשיפור מתמיד של תהליכי עבודה חכמים. באמצעות ניתוח נתונים היסטוריים, זיהוי דפוסים ומדידת האפקטיביות של החלטות הבינה המלאכותית, מפתחים יכולים לשפר ולשכלל באופן איטרטיבי את לוגיקת תזמור תהליכי העבודה.

שיקולים ושיטות עבודה מומלצות

בעת יישום ניטור ורישום בתזמור תהליכי עבודה חכמים, יש לשקול את שיטות העבודה המומלצות הבאות:

1. **הגדרת מדדי ניטור ברורים:** זהו את המדדים והאירועים העיקריים שיש לנטר בהתבסס על הדרישות הספציפיות של תהליך העבודה. התמקדו במדדים המספקים תובנות משמעותיות לגבי ביצועי המערכת, בריאותה והתנהגותה.

2. **יישום רישום מפורט:** ודאו כי הצהרות רישום ממוקמות בנקודות המתאימות בתוך רכיבי תהליך העבודה ונקודות ההחלטה של הבינה המלאכותית. תעדו מידע הקשרי רלוונטי, כגון פרמטרים של קלט, תוצאות פלט וכל נתון ביניים שנוצר.

3. **שימוש ברישום מובנה:** אמצו פורמט רישום מובנה כדי להקל על ניתוח ועיבוד נתוני היומן. רישום מובנה מאפשר יכולת חיפוש, סינון וקיבוץ טובים יותר של רשומות היומן.

4. **ניהול שמירת וסיבוב יומנים:** יישמו מדיניות שמירה וסיבוב יומנים כדי לנהל את האחסון ומחזור החיים של קבצי היומן. קבעו את תקופת השמירה המתאימה בהתבסס על דרישות חוקיות, אילוצי אחסון וצרכי ניתוח. אם אפשר, העבירו את הרישום לשירות צד שלישי כמו Papertrail.

5. **אבטחת מידע רגיש:** יש לנקוט זהירות בעת תיעוד מידע רגיש, כגון מידע מזהה אישי או מידע עסקי חסוי. יש ליישם אמצעי אבטחה מתאימים, כגון הסתרת מידע או הצפנה, כדי להגן על מידע רגיש בקבצי התיעוד.

6. **שילוב עם כלי ניטור והתראות:** יש למנף כלי ניטור והתראות לריכוז האיסוף, הניתוח והויזואליזציה של נתוני הניטור והתיעוד. כלים אלה יכולים לספק תובנות בזמן אמת, ליצור התראות על בסיס ספי סף מוגדרים מראש, ולסייע בזיהוי ופתרון בעיות באופן יזום. הכלי המועדף עליי הוא Datadog.

באמצעות יישום מנגנוני ניטור ותיעוד מקיפים, מפתחים יכולים לקבל תובנות חשובות לגבי ההתנהגות והביצועים של תהליכי עבודה חכמים. תובנות אלה מאפשרות ניפוי באגים יעיל, אופטימיזציה ושיפור מתמשך של מערכות תזמור תהליכי עבודה מבוססות בינה מלאכותית.

שיקולי יכולת הרחבה וביצועים

יכולת הרחבה וביצועים הם היבטים קריטיים שיש לקחת בחשבון בעת תכנון ויישום מערכות תזמור תהליכי עבודה חכמים. ככל שהיקף תהליכי העבודה המקבילים ומורכבות הרכיבים מבוססי הבינה המלאכותית גדלים, חיוני להבטיח שהמערכת תוכל לטפל בעומס ביעילות ולהתרחב באופן חלק כדי לענות על הדרישות הגדלות.

טיפול בנפח גבוה של תהליכי עבודה מקבילים

מערכות תזמור תהליכי עבודה חכמים נדרשות לעתים קרובות לטפל במספר רב של תהליכי עבודה מקבילים. כדי להבטיח יכולת הרחבה, יש לשקול את האסטרטגיות הבאות:

1. **עיבוד אסינכרוני:** יישום מנגנוני עיבוד אסינכרוני כדי לנתק את הביצוע של רכיבי תהליך העבודה. זה מאפשר למערכת לטפל במספר תהליכי עבודה במקביל מבלי לחסום או להמתין להשלמת כל רכיב. ניתן להשיג עיבוד אסינכרוני באמצעות תורי הודעות, ארכיטקטורה מונעת אירועים, או מסגרות עבודה לעיבוד משימות רקע כגון Sidekiq.

2. **ארכיטקטורה מבוזרת:** תכנון ארכיטקטורת המערכת כך שתשתמש ברכיבים ללא שרת (כגון AWS Lambda) או פשוט תפזר את העומס בין מספר צמתים או שרתים לצד שרת האפליקציה הראשי. זה מאפשר יכולת הרחבה אופקית, כאשר ניתן להוסיף צמתים נוספים כדי לטפל בנפח מוגבר של תהליכי עבודה.

3. **ביצוע מקבילי:** זיהוי הזדמנויות לביצוע מקבילי בתוך תהליכי העבודה. חלק מרכיבי תהליך העבודה עשויים להיות בלתי תלויים זה בזה וניתן לבצע אותם במקביל. באמצעות שימוש בטכניקות עיבוד מקבילי, כגון ריבוי תהליכים או תורי משימות מבוזרים, המערכת יכולה למטב את ניצול המשאבים ולהפחית את זמן הביצוע הכולל של תהליך העבודה.

אופטימיזציה של ביצועי רכיבים מבוססי בינה מלאכותית

רכיבים מבוססי בינה מלאכותית, כגון מודלים של למידת מכונה או מנועי עיבוד שפה טבעית, עשויים להיות עתירי משאבי מחשוב ולהשפיע על הביצועים הכוללים של מערכת תזמור תהליכי העבודה. כדי לייעל את ביצועי רכיבי הבינה המלאכותית, שקלו את הטכניקות הבאות:

1. **מטמון:** אם עיבוד הבינה המלאכותית שלכם הוא גנרטיבי טהור ואינו כולל חיפושי מידע בזמן אמת או אינטגרציות חיצוניות כדי לייצר את השלמות הצ'אט, תוכלו לבחון מנגנוני מטמון לאחסון ושימוש חוזר בתוצאות של פעולות שנגישות תכופות או עתירות משאבי מחשוב.

2. **אופטימיזציה של מודלים:** יש לבצע אופטימיזציה מתמדת של אופן השימוש במודלים של בינה מלאכותית ברכיבי תהליך העבודה. זה עשוי לכלול טכניקות כמו זיקוק פרומפטים או פשוט להיות עניין של בדיקת מודלים חדשים כשהם הופכים זמינים.

3. **עיבוד אצווה:** אם אתם עובדים עם מודלים מסוג GPT-4, ייתכן שתוכלו לנצל טכניקות עיבוד אצווה כדי לעבד מספר נקודות מידע או בקשות באצווה אחת, במקום לעבד אותן בנפרד. על ידי עיבוד נתונים באצוות, המערכת יכולה לייעל את ניצול המשאבים ולהפחית את התקורה של בקשות מודל חוזרות.

ניטור ופרופיילינג ביצועים

כדי לזהות צווארי בקבוק בביצועים ולייעל את הסקילביליות של מערכת תזמור תהליכי העבודה החכמה, חיוני ליישם מנגנוני ניטור ופרופיילינג. שקלו את הגישות הבאות:

1. **מדדי ביצועים:** הגדירו ועקבו אחר מדדי ביצועים מרכזיים, כגון זמן תגובה, תפוקה, ניצול משאבים והשהיה. מדדים אלה מספקים תובנות לגבי ביצועי המערכת ועוזרים לזהות תחומים לאופטימיזציה. מאגד מודלי הבינה המלאכותית הפופולרי OpenRouter כולל מדדי Host[1] ו-Speed[2] בכל תגובת API, מה שהופך את המעקב אחר מדדי מפתח אלה לפשוט.

---

[1] Host הוא הזמן שלקח לקבל את הבייט הראשון של הדור המוזרם ממארח המודל, או במילים אחרות "זמן עד לבייט הראשון."

[2] Speed מחושב כמספר אסימוני ההשלמה מחולק בזמן הייצור הכולל. עבור בקשות שאינן מוזרמות, ההשהיה נחשבת כחלק מזמן הייצור.

2. **כלי פרופיילינג:** השתמשו בכלי פרופיילינג כדי לנתח את הביצועים של רכיבי תהליך העבודה הבודדים ופעולות הבינה המלאכותית. כלי פרופיילינג יכולים לעזור בזיהוי נקודות חמות בביצועים, נתיבי קוד לא יעילים, או פעולות עתירות משאבים. כלי פרופיילינג פופולריים כוללים את New Relic, Scout, או כלי פרופיילינג מובנים המסופקים על ידי שפת התכנות או המסגרת.

3. **בדיקות עומס:** ביצוע בדיקות עומס להערכת ביצועי המערכת תחת רמות שונות של עומסים מקבילים. בדיקות עומס מסייעות בזיהוי מגבלות הסילום של המערכת, באיתור ירידה בביצועים, ובהבטחה שהמערכת יכולה לטפל בתעבורה הצפויה מבלי לפגוע בביצועים.

4. **ניטור מתמשך:** יישום מנגנוני ניטור והתראה מתמשכים כדי לזהות באופן פרואקטיבי בעיות ביצועים וצווארי בקבוק. הגדרת לוחות מחוונים והתראות לניטור מדדי ביצוע עיקריים (KPIs) וקבלת התראות כאשר חורגים מספי הביצוע שהוגדרו מראש. זה מאפשר זיהוי ופתרון מהיר של בעיות ביצועים.

אסטרטגיות סילום

כדי לטפל בעומסי עבודה גדלים ולהבטיח את יכולת הסילום של מערכת תזמור תהליכי העבודה החכמה, יש לשקול את אסטרטגיות הסילום הבאות:

1. **סילום אנכי:** סילום אנכי כולל הגדלת המשאבים (למשל, מעבד, זיכרון) של צמתים או שרתים בודדים כדי לטפל בעומסי עבודה גבוהים יותר. גישה זו מתאימה כאשר המערכת דורשת יותר כוח עיבוד או זיכרון כדי לטפל בתהליכי עבודה מורכבים או בפעולות בינה מלאכותית.

2. **סילום אופקי:** סילום אופקי כולל הוספת צמתים או שרתים נוספים למערכת כדי לחלק את העומס. גישה זו יעילה כאשר המערכת צריכה לטפל במספר רב של תהליכי עבודה מקבילים או כאשר ניתן לחלק בקלות את העומס בין מספר צמתים. סילום אופקי דורש ארכיטקטורה מבוזרת ומנגנוני איזון עומסים כדי להבטיח חלוקה שווה של התעבורה.

3. **סילום אוטומטי:** יישום מנגנוני סילום אוטומטי כדי להתאים באופן אוטומטי את מספר הצמתים או המשאבים בהתאם לדרישות העומס. סילום אוטומטי מאפשר למערכת להתרחב או להצטמצם באופן דינמי בהתאם לתעבורה הנכנסת, תוך הבטחת ניצול מיטבי של משאבים

ויעילות עלויות. פלטפורמות ענן כמו Amazon Web Services (AWS) או Google Cloud Platform (GCP) מספקות יכולות סילום אוטומטי שניתן לנצל עבור מערכות תזמור תהליכי עבודה חכמות.

טכניקות לאופטימיזציית ביצועים

בנוסף לאסטרטגיות הסילום, יש לשקול את טכניקות אופטימיזציית הביצועים הבאות כדי לשפר את היעילות של מערכת תזמור תהליכי העבודה החכמה:

1. **אחסון ואחזור נתונים יעיל:** אופטימיזציה של מנגנוני אחסון ואחזור הנתונים המשמשים את רכיבי תהליך העבודה. שימוש באינדוקס יעיל של בסיס הנתונים, טכניקות אופטימיזציה של שאילתות, ומטמון נתונים כדי למזער את זמני התגובה ולשפר את הביצועים של פעולות עתירות נתונים.

2. **קלט/פלט א-סינכרוני:** שימוש בפעולות קלט/פלט א-סינכרוניות כדי למנוע חסימה ולשפר את זמני התגובה של המערכת. קלט/פלט א-סינכרוני מאפשר למערכת לטפל במספר בקשות במקביל מבלי להמתין להשלמת פעולות קלט/פלט, ובכך למקסם את ניצול המשאבים.

3. **סריאליזציה ודסריאליזציה יעילה:** ייעול תהליכי הסריאליזציה והדסריאליזציה המשמשים להחלפת נתונים בין רכיבי תהליך העבודה. שימוש בפורמטים יעילים לסריאליזציה, כגון Buffers Protocol או MessagePack, כדי להפחית את העומס של סריאליזציית נתונים ולשפר את ביצועי התקשורת בין הרכיבים.

עבור יישומים מבוססי Ruby, שקול להשתמש ב-ID Universal. ID Universal משתמש הן ב-MessagePack והן ב-Brotli (שילוב שנבנה למהירות ודחיסת נתונים מהטובות בתחום). כאשר משולבים יחד, ספריות אלה מהירות עד 30% ומגיעות לשיעורי דחיסה הקרובים ב-2-5% ל-Buffers. Protocol

4. **דחיסה וקידוד:** יישום טכניקות דחיסה וקידוד להקטנת נפח הנתונים המועברים בין רכיבי תהליך העבודה. אלגוריתמי דחיסה, כגון gzip או Brotli, יכולים להפחית משמעותית את השימוש ברוחב פס הרשת ולשפר את הביצועים הכוללים של המערכת.

באמצעות התחשבות בהיבטי הרחבה וביצועים במהלך התכנון והיישום של מערכות תזמור תהליכי עבודה חכמות, ניתן להבטיח שהמערכת שלך תוכל לטפל בנפח גבוה של תהליכי

עבודה מקבילים, לייעל את ביצועי הרכיבים מבוססי הבינה המלאכותית, ולהתרחב בצורה חלקה כדי לעמוד בדרישות הגדלות. ניטור מתמשך, פרופיילינג ומאמצי אופטימיזציה הם חיוניים לשמירה על ביצועי המערכת והיענותה ככל שהעומס והמורכבות גדלים לאורך זמן.

בדיקות ותיקוף של תהליכי עבודה

בדיקות ותיקוף הם היבטים קריטיים בפיתוח ותחזוקה של מערכות תזמור תהליכי עבודה חכמות. בהתחשב באופי המורכב של תהליכי עבודה מבוססי בינה מלאכותית, חיוני להבטיח שכל רכיב פועל כמצופה, שתהליך העבודה הכולל מתנהג כראוי, ושההחלטות המבוססות על בינה מלאכותית מדויקות ואמינות. בחלק זה, נחקור טכניקות ושיקולים שונים לבדיקה ותיקוף של תהליכי עבודה חכמים.

בדיקות יחידה לרכיבי תהליך העבודה

בדיקות יחידה כוללות בדיקת רכיבי תהליך עבודה בודדים בבידוד כדי לוודא את נכונותם ועמידותם. בעת ביצוע בדיקות יחידה לרכיבי תהליך עבודה מבוססי בינה מלאכותית, יש להתחשב בנקודות הבאות:

1. **אימות קלט:** בדיקת יכולת הרכיב לטפל בסוגים שונים של קלטים, כולל נתונים תקינים ולא תקינים. יש לוודא שהרכיב מטפל בצורה הולמת במקרי קצה ומספק הודעות שגיאה או חריגים מתאימים.

2. **אימות פלט:** וידוא שהרכיב מייצר את הפלט הצפוי עבור קבוצת קלטים נתונה. השוואת הפלט בפועל עם התוצאות הצפויות כדי להבטיח נכונות.

3. **טיפול בשגיאות:** יש לבדוק את מנגנוני הטיפול בשגיאות של הרכיב על ידי הדמיה של תרחישי שגיאה שונים, כגון קלט לא תקין, אי-זמינות משאבים, או חריגות בלתי צפויות. יש לוודא שהרכיב לוכד ומטפל בשגיאות בצורה נאותה.

4. **תנאי קצה:** יש לבדוק את התנהגות הרכיב תחת תנאי קצה, כגון קלט ריק, גודל קלט מקסימלי, או ערכים קיצוניים. יש לוודא שהרכיב מטפל בתנאים אלה בצורה חלקה מבלי לקרוס או לייצר תוצאות שגויות.

להלן דוגמה לבדיקת יחידה עבור רכיב תהליך עבודה ב-Ruby באמצעות מסגרת הבדיקות RSpec:

```ruby
RSpec.describe OrderValidator do
  describe '#validate' do
    context 'valid is order when' do
      let(:order) { build(:order) }

      it 'true returns' do
        expect(subject.validate(order)).to be true
      end
    end

    context 'invalid is order when' do
      let(:order) { build(:order, total_amount: -100) }

      it 'false returns' do
        expect(subject.validate(order)).to be false
      end
    end
  end
end
```

בדוגמה זו, הרכיב OrderValidator נבדק באמצעות שני מקרי בדיקה: אחד עבור הזמנה תקינה ואחד עבור הזמנה לא תקינה. מקרי הבדיקה מוודאים שהמתודה validate מחזירה את הערך הבוליאני הצפוי בהתאם לתקינות ההזמנה.

בדיקות אינטגרציה של אינטראקציות בתזרים העבודה

בדיקות אינטגרציה מתמקדות באימות האינטראקציות וזרימת הנתונים בין רכיבים שונים בתזרים העבודה. הן מבטיחות שהרכיבים עובדים יחד בצורה חלקה ומפיקים את התוצאות הצפויות. בעת ביצוע בדיקות אינטגרציה לתזרימי עבודה חכמים, יש להתחשב בנקודות הבאות:

1. **אינטראקציה בין רכיבים:** בדיקת התקשורת וחילופי הנתונים בין רכיבי תזרים העבודה. אימות שהפלט של רכיב אחד מועבר כקלט לרכיב הבא בתזרים העבודה בצורה נכונה.

2. **עקביות נתונים:** וידוא שהנתונים נשארים עקביים ומדויקים בזרימתם דרך תזרים העבודה. אימות שהמרות נתונים, חישובים וצבירות מתבצעים בצורה נכונה.

3. **התפשטות חריגים:** בדיקת האופן שבו חריגים ושגיאות מתפשטים ומטופלים בין רכיבי תזרים העבודה. אימות שחריגים נתפסים, נרשמים ומטופלים כראוי כדי למנוע שיבוש בתזרים העבודה.

4. **התנהגות אסינכרונית:** אם תזרים העבודה כולל רכיבים אסינכרוניים או ביצוע מקבילי, יש לבדוק את מנגנוני התיאום והסנכרון. וידוא שתזרים העבודה מתנהג כראוי בתרחישים מקביליים ואסינכרוניים.

להלן דוגמה לבדיקת אינטגרציה עבור תזרים עבודה ב-Ruby באמצעות מסגרת הבדיקות RSpec:

```ruby
RSpec.describe OrderProcessingWorkflow do

  let(:order) { build(:order) }

  it 'successfully order the processes' do
    expect(OrderValidator).to receive(:validate).and_return(true)
    expect(InventoryManager).to receive(:check_availability).and_return(true)
    expect(PaymentProcessor).to receive(:process_payment).and_return(true)
    expect(ShippingService).to receive(:schedule_shipping).and_return(true)

    workflow = OrderProcessingWorkflow.new(order)
    result = workflow.process

    expect(result).to be true
    expect(order.status).to eq('processed')
  end

end
```

בדוגמה זו, ה-OrderProcessingWorkflow נבדק על ידי אימות האינטראקציות בין רכיבי זרימת העבודה השונים. מקרה הבדיקה מגדיר ציפיות להתנהגות של כל רכיב ומוודא שזרימת העבודה מעבדת את ההזמנה בהצלחה, תוך עדכון סטטוס ההזמנה בהתאם.

בדיקת נקודות החלטה של בינה מלאכותית

בדיקת נקודות החלטה של בינה מלאכותית היא קריטית להבטחת הדיוק והאמינות של זרימות עבודה מבוססות בינה מלאכותית. בעת בדיקת נקודות החלטה של בינה מלאכותית, יש לקחת בחשבון את הנקודות הבאות:

1. **דיוק החלטות:** יש לוודא שרכיב הבינה המלאכותית מקבל החלטות מדויקות בהתבסס על נתוני הקלט והמודל המאומן. יש להשוות את החלטות הבינה המלאכותית לתוצאות הצפויות או לנתוני האמת.

2. **מקרי קצה:** יש לבדוק את התנהגות רכיב הבינה המלאכותית במקרי קצה ותרחישים חריגים. יש לוודא שרכיב הבינה המלאכותית מטפל במקרים אלה בצורה חלקה ומקבל החלטות סבירות.

3. **הטיה והוגנות:** יש להעריך את רכיב הבינה המלאכותית לאיתור הטיות פוטנציאליות ולוודא שהוא מקבל החלטות הוגנות וחסרות פניות. יש לבדוק את הרכיב עם נתוני קלט מגוונים ולנתח את התוצאות לאיתור דפוסים מפלים.

4. **יכולת הסבר:** אם רכיב הבינה המלאכותית מספק הסברים או נימוקים להחלטותיו, יש לוודא את נכונות ובהירות ההסברים. יש להבטיח שההסברים תואמים את תהליך קבלת ההחלטות הבסיסי.

להלן דוגמה לבדיקת נקודת החלטה של בינה מלאכותית ב-Ruby באמצעות מסגרת הבדיקות RSpec:

```
RSpec.describe FraudDetector do
  describe '#detect_fraud' do
    context 'fraudulent is transaction when' do
      let(:tx) do
        build(:transaction, amount: 10_000, location: 'Country High-Risk')
      end

      it 'true returns' do
        expect(subject.detect_fraud(tx)).to be true
      end
    end

    context 'legitimate is transaction when' do
      let(:tx) do
        build(:transaction, amount: 100, location: 'Country Low-Risk')
      end

      it 'false returns' do
        expect(subject.detect_fraud(tx)).to be false
      end
    end
  end
end
```

בדוגמה זו, רכיב ה-AI FraudDetector נבדק באמצעות שני מקרי בדיקה: אחד עבור עסקה מזויפת ואחד עבור עסקה לגיטימית. מקרי הבדיקה מוודאים שהמתודה fraud_detect מחזירה את הערך הבוליאני הצפוי בהתבסס על מאפייני העסקה.

בדיקות מקצה לקצה

בדיקות מקצה לקצה כוללות בדיקה של כל תהליך העבודה מההתחלה ועד הסוף, תוך הדמיית תרחישים מהעולם האמיתי ואינטראקציות משתמש. הן מבטיחות שתהליך העבודה מתנהג כראוי ומפיק את התוצאות הרצויות. בעת ביצוע בדיקות מקצה לקצה עבור תהליכי עבודה חכמים, יש לקחת בחשבון את הנקודות הבאות:

1. **תרחישי משתמש:** זיהוי תרחישי משתמש נפוצים ובדיקת התנהגות תהליך העבודה בתרחישים אלה. יש לוודא שתהליך העבודה מטפל בקלט המשתמש כראוי, מקבל החלטות מתאימות ומפיק את הפלט הצפוי.

2. **אימות נתונים:** יש לוודא שתהליך העבודה מאמת ומנקה את קלט המשתמש כדי למנוע חוסר עקביות בנתונים או פרצות אבטחה. יש לבדוק את תהליך העבודה עם סוגים שונים של נתוני קלט, כולל נתונים תקינים ולא תקינים.

3. **התאוששות משגיאות:** בדיקת יכולת תהליך העבודה להתאושש משגיאות וחריגות. יש להדמות תרחישי שגיאה ולוודא שתהליך העבודה מטפל בהם בצורה הולמת, מתעד את השגיאות ונוקט בפעולות התאוששות מתאימות.

4. **ביצועים וסקילביליות:** הערכת הביצועים והסקילביליות של תהליך העבודה תחת תנאי עומס שונים. יש לבדוק את תהליך העבודה עם נפח גדול של בקשות מקבילות ולמדוד זמני תגובה, ניצול משאבים ויציבות כללית של המערכת.

להלן דוגמה לבדיקת מקצה לקצה עבור תהליך עבודה ב-Ruby באמצעות מסגרת הבדיקות RSpec וספריית Capybara להדמיית אינטראקציות משתמש:

```
RSpec.describe 'Workflow Processing Order' do
  scenario 'successfully order an places User' do
    visit '/orders/new'
    fill_in 'Product', with: 'Product Sample'
    fill_in 'Quantity', with: '2'
    fill_in 'Address Shipping', with: 'St Main 123'
    click_button 'Order Place'

    expect(page).to have_content('Successfully Placed Order')
    expect(Order.count).to eq(1)
    expect(Order.last.status).to eq('processed')
  end
end
```

בדוגמה זו, בדיקת הקצה לקצה מדמה משתמש המבצע הזמנה דרך ממשק האינטרנט. היא ממלאת את שדות הטופס הנדרשים, שולחת את ההזמנה, ומוודאת שההזמנה מעובדת בהצלחה, מציגה את הודעת האישור המתאימה ומעדכנת את סטטוס ההזמנה במסד הנתונים.

אינטגרציה ופריסה מתמשכת

כדי להבטיח את האמינות והתחזוקתיות של תהליכי עבודה חכמים, מומלץ לשלב בדיקות ותיקוף בצינור האינטגרציה והפריסה המתמשכת (CI/CD). זה מאפשר בדיקה ותיקוף

אוטומטיים של שינויים בתהליך העבודה לפני פריסתם לסביבת הייצור. שקול את השיטות הבאות:

1. **הרצת בדיקות אוטומטית:** הגדר את צינור ה-CI/CD להריץ אוטומטית את מערך הבדיקות בכל פעם שמתבצעים שינויים בקוד של תהליך העבודה. זה מבטיח שכל רגרסיה או כשל יתגלו מוקדם בתהליך הפיתוח.

2. **ניטור כיסוי בדיקות:** מדוד ונטר את כיסוי הבדיקות של רכיבי תהליך העבודה ונקודות ההחלטה של הבינה המלאכותית. שאף לכיסוי בדיקות גבוה כדי להבטיח שנתיבים ותרחישים קריטיים נבדקים ביסודיות.

3. **משוב מתמשך:** שלב את תוצאות הבדיקות ומדדי איכות הקוד בתהליך הפיתוח. ספק משוב מתמשך למפתחים לגבי מצב הבדיקות, איכות הקוד, ובעיות שהתגלו במהלך תהליך ה-CI/CD.

4. **סביבות קדם-ייצור:** פרוס את תהליך העבודה לסביבות קדם-ייצור המדמות בצורה קרובה את סביבת הייצור. בצע בדיקות ותיקוף נוספים בסביבת הקדם-ייצור כדי לאתר בעיות הקשורות לתשתית, תצורה, או אינטגרציית נתונים.

5. **מנגנוני שחזור:** יישם מנגנוני שחזור למקרה של כשלים בפריסה או בעיות קריטיות שמתגלות בייצור. ודא שניתן לשחזר במהירות את תהליך העבודה לגרסה יציבה קודמת כדי למזער זמני השבתה והשפעה על המשתמשים.

על ידי שילוב בדיקות ותיקוף לאורך מחזור החיים של פיתוח תהליכי עבודה חכמים, ארגונים יכולים להבטיח את האמינות, הדיוק והתחזוקתיות של המערכות מבוססות הבינה המלאכותית שלהם. בדיקות ותיקוף סדירים עוזרים לאתר באגים, למנוע רגרסיות, ולבנות ביטחון בהתנהגות ובתוצאות של תהליך העבודה.

חלק 2: התבניות

הנדסת פרומפטים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

שרשרת חשיבה

כיצד זה עובד

דוגמאות

יצירת תוכן

יצירת ישויות מובנות

הנחיית סוכן LLM

יתרונות ושיקולים

החלפת מצב

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מתי להשתמש בזה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

הקצאת תפקיד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מתי להשתמש בזה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמאות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

אובייקט הנחיה

איך זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

תבנית פרומפט

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יתרונות ושיקולים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מתי להשתמש בזה:

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

קלט/פלט מובנה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

הרחבת קלט/פלט מובנה לעיבוד מקבילי

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יתרונות ושיקולים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

שרשור פרומפטים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מתי להשתמש בזה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה: תהליך החניכה של Olympia

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מעבד הנחיות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

תיחום תגובות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

איך זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יתרונות ושיקולים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

טיפול בשגיאות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מנתח שאילתות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מימוש

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

תיוג חלקי דיבור (POS) וזיהוי ישויות מתויגות (NER)

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

סיווג כוונות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

חילוץ מילות מפתח

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יתרונות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מעבד שאילתות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יתרונות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

Ventriloquist

כיצד זה עובד

מתי להשתמש בזה

דוגמה

רכיבים בדידים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

פרדיקט

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מתי להשתמש בו

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

חזית ממשק תכנות (API) Facade

כיצד זה עובד

יתרונות מרכזיים

מתי להשתמש בו

דוגמה

אימות והרשאה

טיפול בבקשות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

עיצוב תגובות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

טיפול בשגיאות ומקרי קצה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

שיקולי סקילביליות וביצועים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

השוואה עם תבניות עיצוב אחרות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מפרש תוצאות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מתי להשתמש בו

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מכונה וירטואלית

כיצד זה עובד

מתי להשתמש בזה

דוגמה

מאחורי הקסם

אפיון ובדיקות

אפיון ההתנהגות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כתיבת מקרי בדיקה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה: בדיקת רכיב המתרגם

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

הפעלה חוזרת של אינטראקציות HTTP

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

אדם בתוך המעגל (HITL)

תבניות ברמה גבוהה

אינטליגנציה היברידית

תגובה מסתגלת

החלפת תפקידים בין אדם ובינה מלאכותית

הסלמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יתרונות מרכזיים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יישום בעולם האמיתי: שירותי בריאות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

לולאת משוב

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

איך זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יישומים ודוגמאות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

טכניקות מתקדמות בשילוב משוב אנושי

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

הקרנת מידע פסיבית

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

תצוגת מידע הקשרית

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

התראות יזומות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

תובנות מסבירות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

חקירה אינטראקטיבית

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יתרונות מרכזיים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יישומים ודוגמאות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

קבלת החלטות שיתופית (CDM)

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

למידה מתמשכת

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יישומים ודוגמאות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

שיקולים אתיים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

תפקיד המעורבות האנושית במעגל בהפחתת סיכוני בינה מלאכותית

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

התקדמות טכנולוגית ותחזית עתידית

אתגרים ומגבלות של מערכות HITL

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

טיפול חכם בשגיאות

גישות מסורתיות לטיפול בשגיאות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

אבחון שגיאות הקשרי

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

איך זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

הנדסת הנחיות לאבחון שגיאות הקשרי

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

ייצור מועשר באחזור לאבחון שגיאות הקשרי

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דיווח שגיאות חכם

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מניעת שגיאות חזויה

איך זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

התאוששות חכמה משגיאות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

תקשורת שגיאות מותאמת אישית

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

תהליך עבודה מסתגל לטיפול בשגיאות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

איך זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

בקרת איכות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

Eval

בעיה

פתרון

איך זה עובד

דוגמה

שיקולים

הבנת התייחסויות זהב

כיצד פועלות הערכות ללא התייחסות

מעקה בטיחות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

בעיה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

פתרון

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

כיצד זה עובד

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

דוגמה

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

שיקולים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מעקי בטיחות והערכות: שני צדדים של אותו מטבע

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

החלפה הדדית של מעקי בטיחות והערכות ללא התייחסות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

יישום מעקות בטיחות והערכות דו-תכליתיות

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

מילון מונחים

מילון מונחים

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

A

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

B

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

C

תוכן זה אינו זמין בספר הדוגמה. ניתן לרכוש את הספר ב-Leanpub ב-http://leanpub.com/patterns-of-application-development-using-ai-he.

D

E

F

G

H

I

J

K

L

M

N

O

P

ק

ר

S

T

U

V

W

Z

אינדקס