בס"ד

האוניברסיטה הפתוחה
THE OPEN UNIVERSITY OF ISRAEL
الجامعة المفتوحة

סמינר בהנדסת תוכנה (20368)

תכנות באמצעות מודלי שפה גדולים

מגיש: אריה אנקרי (324676683)

מנחה: פרופ' שמואל טישברוביץ

תאריך הגשה: 12/01/2025

תוכן העניינים

1. מבוא

מאז יצא לעולם ChatGPT – מודל הבינה המלאכותית של חברת OpenAI – בשנת 2020, חלה עלייה חדה במספרם ובשימושם של כלים מבוססי בינה מלאכותית. פעולות שנראו עד אז כבלעדיות עבור בני האדם, כגון כתיבת מאמרים, שיחה טבעית וזיהוי פרצופים, נעשו לפתע גם נחלת המחשב. כלי הבינה המלאכותית הקיימים היום מאפשרים ליצור תמונות, לתרגם באופן מלאכותי מאמרים שלמים, לנתח את תוכנם, ולכתוב חדשים באמצעות הקלדת משפט בודד [5]. בין כלי הבינה המלאכותית הפופולאריים ניתן למנות את ChatGPT, Dall-E, OpenCV, TensorFlow, GitHub Copilot ועוד.

קטגוריה נפוצה מאוד של כלי בינה מלאכותית הם מודלי השפה הגדולים (LLM – Large Language Model). כלים אלו הם מבוססי טקסט, כלומר שעיקר תפקידם הוא לייצר טקסט ע"י הקלדת משפט הנחיה המתאר את רצון המשתמש. כלים אלו, ש- ChatGPT הוא המוביל והמוכר שבהם, הראו יכולות מרשימות מאוד בייצור טקסטים מכל תחום.

תחום שנפוץ בו השימוש בכלים אלו הוא תחום התכנות. כלי הבינה המלאכותית הוכיחו את יכולותיהם בכתיבת קוד בכל שפת תכנות. הקוד המיוצר לרוב יכיל מספר מאוד מצומצם של שגיאות (באגים), הן בפן התחבירי והן מבחינת הביצוע של הקוד והעמידה במשימה שהוכתבה. הצלחה זו גרמה למתכנתים רבים להשתמש בכלי הבינה המלאכותית ליצירת תוכנות ותיקון שגיאות. אולם, כמו בני האדם, גם המחשב אינו חף מטעויות. על אף שמספר השגיאות בקוד המיוצר הוא נמוך, עדיין יש לוודא שתוכנה בשימוש מסחרי חפה משגיאות ואכן עושה את המוטל עליה.

בנוסף, היכולת של הבינה המלאכותית ליצור קוד, העמידה קשיים רבים בפני מורים ופרופסורים למדעי המחשב, המעוניינים ללמד את יסודות התכנות. במקרים רבים מורים חששו כי היכולת ליצור קוד באופן אוטומטי תפגע בתהליך הלמידה [1], ואף עלולה להוביל לרמאות בעת ביצוע מטלות בית, הנחוצות לכל סטודנט.

מחקרים רבים בדקו את יכולות התכנות של כלי הבינה המלאכותית, והציבו גבולות בהם ישנו הבדל בין תכנות מעשה ידי אדם לבין תכנות הנעשה ע"י מחשב. מחקרים אלו גם דירגו את איכות הקוד ואף העניקו ציונים [5] לכלים השונים, כדי לתת למתכנתים כלים מתי ניתן לסמוך על הקוד המיוצר ע"י המחשב, ומתי יש לצפות לשגיאות בקוד.

במאמר זה, נבחן את יכולותיהם של שני מודלי שפה גדולים פופולאריים – ChatGPT ו-Gemini – בכתיבת קוד, וננתח את תוצאות הניסוי. בנוסף, נציג את נקודות החוזק והמגבלות של הכלים, ונציע דרכים להבין מתי ניתן לסמוך על קוד המיוצר על ידי בינה מלאכותית ומתי דרושה בדיקה אנושית מעמיקה.

כל קטעי הקוד המופיעים במאמר זה נגישים וניתן למצוא אותם בקישור הבא: https://github.com/AriehA1995/OOP-with-LLM

2. מודלי שפה גדולים

2.1. מהו מודל שפה גדול?

מודל שפה גדול (מאנגלית: Large Language Model ובקיצור LLM) הוא מערכת מבוססת בינה מלאכותית שתפקידה לנתח ולייצר טקסטים במגוון רחב של נושאים ושפות. מודלים אלו מאומנים על מאגרי טקסט עצומים הכוללים ספרים, מאמרים ומסמכים מסוגים שונים. במהלך האימון, המודלים לומדים לזהות מבנים לשוניים, דקדוק, הקשרים סמנטיים ואפילו רגשות של השפה האנושית.

מודל שפה פועל על בסיס תחזיות: הוא מקבל קלט בצורת הנחיה (prompt), מנתח את ההקשר ומנסה לחזות את המילים הבאות שיתאימו להקשר זה. בזכות תהליך הלמידה על כמויות אדירות של נתונים, מודלי שפה גדולים מסוגלים לענות על שאלות במגוון נושאים, לכתוב מאמרים, לתרגם טקסטים, לנהל שיחות טבעיות עם בני אדם ואף לנתח טקסטים קיימים ולספק סיכומים שלהם [5].

מאז יציאת ChatGPT בשנת 2020, הפכו מודלי השפה לכלים מרכזיים בשימוש יומיומי בתחומים מגוונים, כגון: חינוך, טכנולוגיה ועסקים. בין מודלי השפה הפופולאריים כיום ניתן למנות את ChatGPT של חברת OpenAI, Gemini של Google וClaude של Anthropic שכל אחד מהם מציג יכולות ייחודיות ומעט שונות.

2.2. שימוש בעולם התכנות

עולם התכנות לא פסח על השימוש במודלי שפה גדולים. מתכנתים החלו להשתמש במודלי השפה לביצוע מטלות שונות, אשר לרוב מייעלים את העבודה והופכים את עבודת התכנות למהירה יותר.

מודלי השפה מסוגלים לייצר קוד כמעט בכל שפת תכנות. ע"י מתן הנחיה מדויקת, המודלים יכולים לייצר קוד של פונקציות מסויימות, כתיבת אלגוריתמים ואף תוכנות שלמות. בנוסף, המודלים משמשים גם לתחזוקת קוד קיים: הם מסוגלים לזהות באגים, להציע תיקונים ולהסביר קטעי קוד סבוכים.

למרות היתרונות המרשימים, ישנם גם אתגרים. **ראשית**, קוד שנוצר על ידי בינה מלאכותית עלול להכיל שגיאות – תחביריות או סמנטיות – וכן להיות לא יעיל מבחינה ביצועית. **שנית**, המודלים אינם מבינים את הקוד כפי שמתכנתים אנושיים מבינים אותו. הם מבססים את הפלט על סטטיסטיקה והקשרים, לא על הבנה לוגית מעמיקה.

אתגר נוסף עלה **בפן האקדמי**. מורים למדעי המחשב חששו כי סטודנטים יסתמכו על המודלים לפתרון מטלות, ובכך יפגעו בתהליך הלמידה ובהבנה המעמיקה של עקרונות התכנות. כמו כן, כלים אלו עלולים לשמש לרמאות בביצוע עבודות ובחינות בית. מחקרים מצביעים על הצורך בפיתוח שיטות הוראה חדשות שיאפשרו ניצול של יתרונות המודלים תוך שמירה על אתיקה חינוכית [1].

למעשה, השימוש במודלי שפה גדולים בתכנות מציע יתרונות רבים, אך הוא דורש איזון בין אוטומציה לבין בקרה אנושית. כלים אלו משנים את דפוסי העבודה בתעשייה, ומעמידים אתגר משמעותי בפני הקהילה האקדמית. עם שילוב נכון, מודלי שפה יכולים להפוך לכלים חינוכיים

ותעשייתיים רבי ערך. במאמר זה יוצג ניסוי המדגים את היכולות של מודלי השפה הגדולים לצד החסרונות שלהם. בנוסף, נדון כיצד ניתן להשתמש בכלים אלו בצורה מושכלת, הן בתעשייה והן באקדמיה.

3. מחקרים קודמים

מחקרים רבים חקרו את יכולות מודלי השפה בתחום התכנות. המחקרים חקרו את נכונות הקוד המיוצר, איכותו, ואף בדקו את יכולת זיהוי השגיאות של מודלי השפה.

3.1. נכונות ואיכות הקוד

במאמר עדכני [5], מוזכרים מחקרים אשר בדקו את נכונות הקוד המיוצר ע״י מודלי שפה גדולים. המחברים מזכירים שני מדדי הערכה מרכזיים שפותחו – HumanEval, MBPP – ע״מ למדוד פרמטרים אלו בקוד. מדדים אלו מבוססים על מספר טסטים שמבוצעים על הקוד, וכך ניתן לתת ציון על נכונות הקוד. במדדים אלו ישנן שתי בעיות: ראשית, מדדים אלו מבוצעים בשפת Python, ואינם מספקים מענה לנכונות הקוד המיוצר בשפות תכנות אחרות. המחברים מזכירים מספר מדדים שפותחו בעקבות HumanEval ו-MBPP, כדי לתת מענה לבעיה זו. שנית, מדדים אלו ממוקדים לתחום התכנות הפונקציונלי. התכנות הפונקציונלי נועד לתת מענה לבעיה ממוקדת, אך לרוב אינו משמש לבניית אפליקציה שלמה ומורכבת. תחום התכנות מונחה עצמים, הנחשב ליותר מורכב ובעל רכיבים רבים ועקרונות מסויימים, אינו נבדק תחת מדדים אלו.

כמענה לבעיות אלו, פיתחו המחברים מדד הערכה מעודכן, המבסס את הערכתו על עקרונות התכנות מונחה עצמים. גם במדד זה, על מודל השפה ליצור קוד בשפת Python כמענה לשאלה אלגוריתמית, אולם הדרישה מהמודל היא ליצור מחלקות עם תכונות ומתודות העומדות בכל עקרונות התכנות מונחה עצמים, בדגש על הכמסה (Encapsulation) וירושה (Inheritance). לאחר מכן, המחברים בדקו את המדד שלהם על 23 מודלי שפה שונים ונתנו ציון לכל אחד מהם.

מהמחקר עלה, כי על אף שבתחום התכנות הפונקציונלי מודלי השפה הגדולים מצליחים לייצר קוד אשר עומד בדרישות שניתנו, בתחום התכנות מונחה עצמים המודלים כשלו בחלק מהמשימות. מרבית המודלים לא ידעו להגדיר תכונה פרטית בשפת Python, על אף שדרישה זו נאמרה במפורש, וחלק מהם נכשלו בעמידה בדרישות, כגון: מתן שמות מדוייקים לתכונות ומתודות. על פי המחקר, ChatGPT קיבל את הציון הגבוה ביותר מבין מודלי השפה הגדולים שנבדקו, אך גם הוא אינו חף משגיאות בתחום התכנות מונחה עצמים. המחברים מציינים כי המחקר שביצעו אינו מכסה שפות תכנות אחרות, ואינו בודק תחומים מורכבים יותר בתכנות מונחה עצמים כגון העמסה (Overloading) ודריסת מתודות (Overriding).

מסקנות דומות ניתן למצוא במאמר נוסף [1], שכתבו שני מרצים למדעי המחשב מאוניברסיטה בפורטוגל. המאמר מתאר ניסוי הבודק את יכולתו של GPT-3 לענות על מטלות אקדמאיות הניתנות לסטודנטים. הניסוי בוצע בשפה הפורטוגזית ולא באנגלית, משום שהמחברים ביקשו לבדוק את יכולות המודל בשפה שאינה שפת המקור של מרבית מאגרי הנתונים. יתר על כן, הסטודנטים המשתמשים במודל שפה כדי לפתור מטלות ככל הנראה יעתיקו את המטלה כלשונה ולא יטרחו לתרגם אותה. במהלך הניסוי ניתנו לGPT-3 שש מטלות מורכבות מתחום התכנות מונחה עצמים,

בהן על המודל לספק קוד בשפת Java הכולל מחלקות עם תכונות ומתודות, פרטיות וציבוריות, בעלות שמות שהוגדרו באופן מדוייק.

מהניסוי עלה, כי על אף שהמודל מצליח ליצור קוד תקין ברובו, למעט שגיאות קומפילציה שחוזרות על עצמן – כגון אי ייבוא ספריות – המודל אינו עומד בעקרונות התכנות מונחה עצמים. המודל נכשל ביצירת מחלקות מופשטות, מאפשר כפילויות בקוד, ואינו מכיר את ההבדל בין תכונות פרטיות לציבוריות כאשר מדובר בירושה.

המאמר ממליץ למורים למדעי המחשב לשלב את מודלי השפה הגדולים בתוך הכיתה וללמוד וללמד את יתרונותיהם וחסרונותיהם. מרצים יכולים להשתמש במודל כדי ליצור תרגילים או ליצור בדיקות יחידה (unit tests) למטלות. המחברים ממליצים להתאים את המטלות למציאות החדשה, וליצור מטלות מורכבות שאינן ניתנות לפתרון ע"י מתן הנחיה אחת למודל בינה מלאכותית. מטלות אלו צריכות להיות "מבוססות פרויקטים", כוללות הגדרת מחלקות מדוייקות, קשרים מורכבים בין מחלקות והוספת תנאים שאינם טריוויאלים (כגון איסור שימוש ב-instanceof) המקשים על המודל.

במאמר המשך [3], בדקו המחברים את אותן המטלות על GPT-4 וBard. ניסוי זה העלה כי GPT-4 עונה בצורה טובה יותר יחסית לשאר המודלים על הדרישות המורכבות של התכנות מונחה עצמים. במודל זה נמצאו פחות שגיאות והוא אף הצליח לענות על אחת המטלות ללא שגיאות כלל. המחברים חזרו על המלצותיהם למרצים לתת לסטודנטים מטלות מורכבות ומבוססות פרויקטים.

לסיכום, מהמחקרים נראה כי מודלי השפה הגדולים אכן יכולים לייצר קוד עם מספר מועט של שגיאות, אך נכשלים יותר בתחום התכנות מונחה עצמים. מהמחקרים הוכח כי המודלים מסוגלים לתת מענה גם בשפה שאינה שפת המקור של מאגרי הנתונים. למעשה, ניתן לסמוך על המודל כי יכתוב את עיקר הקוד בצורה תקינה, אך עדיין יש לצפות לשגיאות תחביריות או מבניות בקוד.

במאמרי זה אציג ניסוי הדומה לניסוי שתואר לעיל, הבודק את יכולותיהם של ChatGPT וGemini לתת מענה איכותי לבעיות מורכבות בתחום התכנות מונחה עצמים. ביצעתי את הניסוי בשפה העברית ולא באנגלית, מהסיבות שפרטו Alves & Cipriano במאמרם [1].

3.2. זיהוי ותיקון שגיאות

מעבר לכתיבת קוד, מודלי שפה גדולים הוכיחו את עצמם גם ביכולת זיהוי ותיקון שגיאות. הכלי PyDex [6] משתמש במודל שפה Codex שאומן במיוחד על קטעי קוד, על מנת לתקן שגיאות בשפת Python. הכלי מעביר למודל קטע קוד עם שגיאה ומוסיף לו הנחיה לתקן את הקוד.

במאמרם מפרטים Zhang ואחרים את תהליך העבודה של הכלי ואת הבדיקות שעשו עליו. PyDex הצליח לזהות ולתקן 86% מהשגיאות, כאשר חלק מהשגיאות הן תחביריות (Syntax error) וחלק מהן סמנטיות (כלומר, הקוד אינו עומד במשימה שהוגדרה עבורו). כלי זה למעשה מוכיח כי מודלי שפה גדולים יכולים לסייע למתכנתים גם לצורך זיהוי ותיקון שגיאות ולא רק לכתיבת קוד.

3.3. זיהוי שגיאות אוטומטי בתכנות מונחה עצמים

תחום נוסף שעשיתי בו שימוש בניסוי, אך אינו קשור למודלי שפה גדולים, הוא תחום הבדיקות האוטומטיות. בבדיקות יחידה (unit tests), מריצים תוכנית המבצעת את כל הפונקציות בקוד באופן נפרד, וכך בודקת אם חלו שגיאות סמנטיות בפונקציות אלו. בדיקות יחידה נמצאות בשימוש נרחב בתעשיה, אך לרוב לא משמשות לבדיקת תקינות של קוד בתחום התכנות מונחה עצמים.

מאמר בנושא זה [4], מדגים כיצד ניתן באמצעות בדיקות יחידה, לבדוק אם קוד מסויים עומד בעקרונות התכנות מונחה עצמים. המחברים מסבירים את הצורך האקדמי שממנו נבע פיתוח הכלי, הנועד לסייע לסטודנטים להבין את טעויותיהם בתכנות מונחה עצמים ולהשתפר. המאמר מתייחס לשפת Python ומראה כיצד ניתן לבדוק האם שתי מחלקות יורשות ממחלקה אחת, האם תכונה מסויימת יורשת ממחלקה קודמת ועוד.

בניסוי שעשיתי השתמשתי בעקרונות מתוך מאמר זה ע״מ לבדוק את הקוד שיוצר ע״י המודלים ולוודא שהוא אכן עומד בעקרונות התכנות מונחה עצמים.

4. תיאור הניסוי שנעשה

ע״מ לבחון את יכולותיהם של מודלי שפה גדולים, חיברתי ניסוי הבודק מה המענה שהמודלים הפופולאריים כיום יכולים לתת בתחום התכנות. הניסוי בוצע בשפה העברית וכולל הנחיה לכתוב קוד של ספריה בשפת Python העונה לכל דרישות התכנות מונחה עצמים.

לניסוי מספר מטרות, שלא נבדקו במאמרים שהובאו לעיל, או נבדקו באופן חלקי בלבד:

1. בדיקת **איכות הקוד המיוצר ע״י GPT-4 ו-Gemini** – מודלים אלו חדשים יחסית ואחד מהם (Gemini) לא נבדק באף מאמר שהובא לעיל. בניסוי נבדק גם אם קיימות שגיאות תחביריות בקוד (Syntax error) וגם שגיאות סמנטיות.
2. בדיקת המענה שנותנים המודלים **בבעיות מורכבות בתחום תכנות מונחה עצמים** – במאמרים שהובאו לעיל נבדקו בעיות פשוטות מתחום זה. לא נבדק המענה לבעיות מורכבות כגון העמסת ודריסה של מתודות או מחלקה מופשטת. כמו כן, לא נבדקה יכולת המודלים ״לעצב״ את המערכת בעצמם, אלא ניתנו למודל מראש המחלקות הנצרכות. בניסוי זה נוספה הבדיקה של יכולת המודל ״להבין״ את מבנה המערכת ולהוסיף מעצמו מחלקות אב כדי לממש עקרונות חשובים בתכנות מונחה עצמים, כגון ירושה.
3. בדיקת **המענה שנותנים המודלים בשפת Python**. שפת תכנות זו היא אחת הפופולאריות בעולם ויש לבדוק האם גם בשפה זו המענה שניתן הוא איכותי. אמנם באחד המאמרים (מאמר מספר 5) המחקר נעשה בשפה זו, אך לא נבדקו בו בעיות מורכבות מתחום תכנות מונחה עצמים.
4. בדיקת **המענה שנותנים המודלים בשפה העברית**. במאמרים שהובאו לעיל, הבדיקות נעשו בשפות אנגלית ופורטוגזית. מן הראוי לבדוק גם את הצלחת המודלים בשפות אחרות. כפי שפורט באחד המאמרים (מאמר מספר 1), שפת המקור של מרבית מאגרי המידע של המודלים היא אנגלית, כך שייתכן שבעברית ביצועי המודלים יהיו קטנים יותר באיכותם.

אם יתברר כי המודלים יכולים לתת מענה איכותי בעברית, דבר זה יקל מאוד על משתמשים שאינם דוברי אנגלית להשתמש בכלים אלו בצורה טובה.

יש לציין כי הניסוי בוצע על הגרסה החינמית של מודלים אלו (ChatGPT מאפשר שימוש מצומצם במודל GPT-4 בחינם) וייתכן כי הגרסה בתשלום תוביל לתוצאות שונות.

ניתן למצוא את השיחות עם המודלים בקישורים הבאים:

ChatGPT: https://chatgpt.com/share/6746f2a2-834c-8005-ac0e-222ca23b4f1b

Gemini: https://g.co/gemini/share/595d3c9116c0

4.1. הסבר הניסוי

פירוט ההנחיה והמערכת הדרושה

בניסוי ניתנה למודל הנחיה (prompt) בעברית לכתוב קוד של module המייצג ספריה. בהנחיה צויין כי על הקוד להיות בשפת Python וכי עליו לעמוד בארבעת העקרונות של תכנות מונחה עצמים (הפשטה, הכמסה, ירושה ורב צורתיות). ההנחיה נכתבה בצורה מפורטת בדומה למטלות הניתנות לסטודנטים.

לצורך הבדיקות שנעשו על הקוד, היה צורך להגדיר מראש את שמות המחלקות, התכונות והמתודות הקיימות במערכת, מתוך כוונה כי המודל יממש חלקים אלו באמצעות שמות אלו בדיוק.

המערכת כוללת את המחלקות הבאות:

- **לקוח** – מחלקה המייצגת את הלקוח. ללקוח נשמרות התכונות מספר תעודת זהות, שם ומספר טלפון. בנוסף, ללקוח קיימת מתודה __str__ (מתודה ייחודית לPython המגדירה כיצד הדפסת ברירת המחדל של אובייקט) המחזירה מחרוזת בפורמט קריא.
- **ספר** – מחלקה המייצגת ספרים. לספר נשמרות התכונות מזהה, שם, שם מחבר, שנת הוצאה, מחלקה וסטטוס (זמין, מושאל, בתיקון). בנוסף, לספר קיימת תכונה ציבורית קבועה שנקראת Type וצריכה להחזיר את המחרוזת "Book". בהנחיה פורטו מתודות המשמשות לשינוי תכונות הספר וכן מתודה __str__ המחזירה מחרוזת בפורמט קריא המתחילה במילה Book.
- **דיסק** – מחלקה המייצגת דיסקים. לדיסק נשמרות התכונות מזהה, שם, שם זמר, שנת הוצאה, וסטטוס (זמין, מושאל, בתיקון). בנוסף, לדיסק קיימת תכונה ציבורית קבועה שנקראת Type וצריכה להחזיר את המחרוזת "Disk". בהנחיה פורטו מתודות המשמשות לשינוי תכונות הדיסק וכן מתודה __str__ המחזירה מחרוזת בפורמט קריא המתחילה במילה Disk.
- **מגזין** – מחלקה המייצגת מגזינים. למגזין נשמרות התכונות מזהה, שם, שם מוציא לאור, מספר סדרתי, וסטטוס (זמין, מושאל, בתיקון). בנוסף, למגזין קיימת תכונה ציבורית קבועה שנקראת Type וצריכה להחזיר את המחרוזת "Magazine". בהנחיה פורטו מתודות המשמשות לשינוי תכונות המגזין וכן מתודה __str__ המחזירה מחרוזת בפורמט קריא המתחילה במילה Magazine.

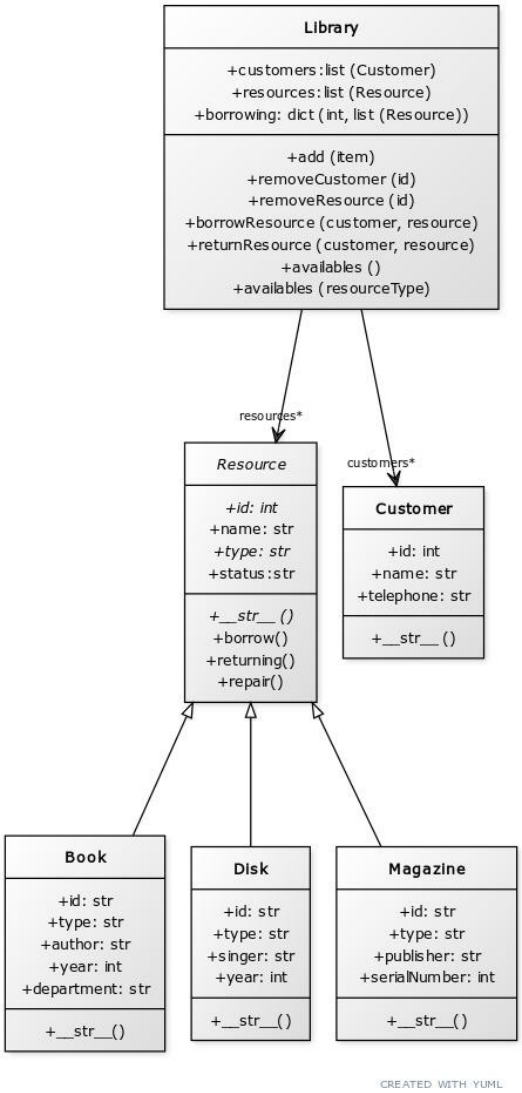
- **ספריה** – המחלקה המייצגת את הספריה. בספריה נשמרות כתכונות פרטיות רשימת הלקוחות, רשימת המשאבים (ספר, דיסק ומגזין) ורשימת השאלות. בנוסף, פורטו בהנחיה מתודות לניהול הספריה כגון הוספה והסרת לקוחות ומשאבים, השאלת והחזרת משאבים והצגת משאבים זמינים.

לכל מחלקות אלו הוגדרו בנאים מתאימים וכן נוספו דרישות לתנאים מסויימים בתכונות, המאלצות את המערכת לבדוק קלט ולהחזיר שגיאות בהתאם. כמו כן, פורטו רשימת התכונות הפרטיות שלא ניתנות לשינוי ורשימת getters וsetters. לתכונות סטטוס במחלקות המשאבים וכן לשלוש התכונות הפרטיות של הספריה הוגדר שאינן ניתנות לשינוי ע"י setters אלא רק באמצעות המתודות הרלוונטיות.

בנוסף, נכתבה בהנחיה הוראה להוסיף קוד הממש מחלקות אלו באמצעות יצירת ספריה, הוספת שלושה משאבים ולקוח אחד. הלקוח ישאל מהספריה את אחד המשאבים, ולבסוף, התוכנית תדפיס את המשאבים הזמינים בספריה לאחר ההשאלה.

בהנחיה הודגש שעל הקוד לעמוד בכל עקרונות תכנות מונחה עצמים ושניתן להוסיף מחלקות או מתודות עזר על מנת לעמוד בדרישה זו. בנוסף, הודגשה חשיבות השמירה על שמות התכונות והמתודות כמות שהם.

יש להדגיש כי בהנחיה לא נכתב כלל אודות מחלקה מסוג **משאב**. ע"פ מבנה הנתונים והחזרתיות במחלקות ספר, דיסק ומגזין, על המודל "להבין" כי ישנו צורך לממש מחלקת משאב על מנת לעמוד בעקרונות התכנות מונחה עצמים וליצור ירושה בין המחלקות ספר, דיסק ומגזין לבין מחלקת האב – משאב. על מחלקת המשאב להיות מחלקה מופשטת, משום שאין צורך לאתחל אובייקט מסוג משאב במערכת.



איור 1: דיאגרמת UML של מערכת הספריה, כפי שאמורה להיות. התכונות המופיעות כאן הן התכונות הציבוריות הממומשות ע"י getters. לחלק מהתכונות לא הוגדרו setters אלא ניתנות לשינוי רק דרך המתודות.

איור 1 מראה דיאגרמת UML המייצגת את המערכת, כפי שאמורה להיות. יש לציין כי המזהה של המשאבים בפורמט שונה בכל סוג משאב, ולכן הוא מופיע במחלקה משאב בתור תכונה מופשטת וגם במחלקות היורשות ממנה.

בסעיפים הבאים נראה כיצד באות לידי מימוש כל ארבעת העקרונות של התכנות מונחה עצמים במערכת זו וכיצד מצופה מהמודל להתמודד עם דרישה זו.

הפשטה (Abstraction)

למעשה, מרבית עבודת ההפשטה כבר נעשתה במתן ההנחיות ופירוט התכונות של המערכת. הגדרת המחלקות השונות ותכונותיהן כבר בוצעו בזמן הגדרת המערכת ואם המודל אינו חורג מהגדרות אלו, הוא יעמוד היטב בדרישות ההפשטה. אמנם, ניתן להוסיף גם לעקרון ההפשטה את בדיקות תקינות הקלט והחזרת שגיאות במקום המתאים. על המודל לבנות את המערכת כך שתהיה קרובה כמה שיותר למציאות ולהיגיון. למשל, מצופה מהמערכת להחזיר שגיאה בעת השאלת ספר שכבר מושאל ע״י מישהו אחר, למרות שדרישה זו הוזכרה רק בקצרה. מודל אשר יצור קוד הבודק מקרים מעין אלו ייחשב כעומד בעקרון ההפשטה.

הכמסה (Encapsulation)

בשביל לעמוד בעקרון ההכמסה, על הקוד לשמור על תכונות האובייקטים כפרטיות ולהגדיר מתודות להחזרת הערך (getters) ולהגדרה (setters) של תכונות. כלומר, משתמש חיצוני המשתמש בקוד בתור ספריה חיצונית (module) לא יוכל לשנות את ערכי התכונות בצורה ידנית.

אמנם, בשפת Python לא קיימת אפשרות למנוע גישה מוחלטת לתכונות פרטיות. הדרך המקובלת להגדיר תכונה כפרטית היא על ידי הוספת קו תחתון כפול (__) בראש השם, אך עדיין ניתן לשנות את ערך התכונה בצורה עקיפה (לא ניתן לשנות בצורה ישירה, אך ניתן באמצעות הקידומת _ClassName__). כמו כן, הגדרות getters נעשות באמצעות הוספת סמל @property לפני מתודה. נושא יצירת תכונות פרטיות היווה מכשול משמעותי במחקרים שהובאו לעיל [5], שהעלו כי הידע של מודלי השפה בתחום זה הינו מועט. יש לציין כי בהנחיה הובאו שמות התכונות הפרטיות בתוספת הקידומת של תכונה פרטית.

על מנת שהקוד יעמוד בעקרון ההכמסה, מצופה שהקוד שנוצר יכלול את התכונות הפרטיות עם הקידומת המונעת עריכה בצורה ישירה וכי התכונות ניתנות לקבלה והגדרה באמצעות הוספת @property כמקובל בשפת Python.

ירושה (Inheritance)

כפי שצויין לעיל, לא הוזכרה כלל המחלקה **משאב** בהנחיה למודל. דבר זה נעשה על מנת לאתגר את המודל ולראות אם הוא מצליח ״להבין״ דפוסים של מחלקות וליצור מחלקת אב עבור מחלקות אלו. אתגר זה משמעותי מאוד ויכול להראות עד כמה המודל רק ״מבצע הוראות״ או שגם מנסה לשפר את הקוד ולייעל את תהליך כתיבתו. ברור שהוספת מחלקת המשאב קריטית לשמירה על עקרון הירושה וגוררת גם שמירה על עקרון הרב צורתיות כפי שיובא בהמשך. הוספת מחלקת משאב תחסוך כתיבה מיותרת במחלקות היורשות ותהפוך את כל ניהול המשאבים לממשק אחיד.

יתר על כן, מחלקת המשאב צריכה להיות מחלקה מופשטת. הסיבה היא, כי אין צורך ומשמעות להגדרת אובייקט מסוג משאב מבלי להגדיר בדיוק איזה סוג משאב הוא. כמו כן, התכונות id, type והמתודה __str__ גם צריכות להיות מופשטות (כלומר, דרישה שיהיה תכונה או מתודה כזאת במחלקה יורשת, אך ללא מימוש בפועל של המתודה במחלקת האב). אך יש לציין כי לא מוגדרת בשפת Python דרך סטנדרטית להגדיר מחלקה מופשטת. הדרך המקובלת היא באמצעות הספריה abc, אך ניתן לממש זאת גם בדרכים אחרות, מורכבות יותר.

על מנת לעמוד בעקרון הירושה, על הקוד לכלול מחלקת אב (לא חייבת להיקרא בשם Resource) למחלקות ספר, דיסק ומגזין. מחלקת האב צריכה להיות מופשטת ולהגדיר את תכונות המזהה, הסוג ומתודת החזרת המחרוזת כמופשטות.

רב צורתיות (Polymorphism)

רב צורתיות פירושה האפשרות לבצע פעולות דומות על אובייקטים שונים. מקובל לחלק את עקרון הרב צורתיות לשני חלקים, המשלימים זה את זה: דריסת (overriding) והעמסת (overloading) מתודות.

דריסת מתודות פירושה שמחלקה יורשת יכולה לשנות (=לדרוס) מתודה שהגדירה מחלקת האב. במערכת שלנו, המתודה __str__ והתכונה type (שהוגדרה כתכונה קבועה) הן מתודות שנדרסות ע"י המחלקה היורשת, שכן גם אם במחלקת האב יוגדרו מתודות אלו, המחלקות היורשות צריכות לדרוס אותן משום שהערך המוחזר שונה בין מחלקה למחלקה (שכן לכל מחלקה הוגדר type שונה ופורמט שונה למחרוזת ברירת המחדל). למותר לציין כי אם המודל לא יעמוד בעקרון הירושה (כלומר, לא יוסיף מעצמו מחלקת משאב), המודל לא יוכל לממש דריסת מתודות.

העמסת מתודות פירושה שניתן להגדיר מתודה, כך שתתנהג באופן שונה בהתאם לפרמטרים שהיא מקבלת. במערכת שלנו הוגדרו שתי מתודות הכוללות העמסה, שתיהן במחלקה ספריה: add – הוספת משאב או לקוח לספריה, availables – החזרת רשימת המשאבים הפנויים הקיימים בספריה. המתודה הראשונה תתנהג שונה אם תקבל לקוח או משאב כפרמטר, המתודה השניה תחזיר את רשימת כל המשאבים הפנויים הקיימים בספריה, אלא אם כן הוזן כפרמטר סוג המשאב, ואז תחזיר רק את רשימת המשאבים הפנויים מסוג זה.

רב צורתיות מאפשרת גמישות בקוד וחיסכון בכתיבה. הקוד שנוצר יעמוד בעיקרון זה אם יממש דריסה והעמסת מתודות כפי שפורט. יש לציין כי דרישות אלו צויינו במפורש בהנחיה שניתנה.

אופן ביצוע הבדיקות

נעשו מספר פעולות על מנת לבדוק את תקינות ואיכות הקוד הנוצר ע"י המודלים. ראשית, לאחר כתיבת כל הקוד ע"י המודל, כולל חלק ההרצה בסוף הקוד, הרצתי את הקוד ע"מ לבדוק שלא נופלות בו שגיאות תחביריות. במידה ויש שגיאות, השגיאה מוחזרת למודל והוא מתקן את הקוד וחוזר חלילה עד שהקוד יוצא ללא שגיאות.

השלב השני הוא בדיקות יחידה (unit test). כתבתי 24 בדיקות יחידה הבודקות את כל התכונות והמתודות שפורטו, כולל בדיקת החזרת שגיאות ותקינות ערכים. ע"מ לבדוק את עקרונות הירושה והרב צורתיות, השתמשתי בבדיקות דומות לאלו שתוארו במאמר לעיל [4]. בדיקות אלו אפשרו לי לדעת אם מומשה מחלקת אב למשאבים וכן אם בוצעה דריסת מתודות. הרצתי את הבדיקות על הקוד, וכך ניתן לתת ציון 0-24 לקוד שנוצר.

ניתן למצוא את קוד הבדיקות, כמו את קטעי הקוד שנוצרו ע"י המודלים, בקישור הבא: https://github.com/ArielhA1995/OOP-with-LLM

4.2. תוצאות הניסוי

הניסוי העלה כי קיימים פערים משמעותיים בין שני המודלים, לכן אסכם כל אחד מהם בנפרד.

ChatGPT

המענה שChatGPT נתן למשימה היה מאוד מפתיע לטובה. המודל החזיר לאחר ההנחיה את הקוד של הספריה גמור, ושום שינוי לא נצרך בקוד. הקוד היה שלם, כולל את כל המרכיבים הדרושים ורץ ללא שגיאות.

הרצת הבדיקות על הקוד העניקה ציון של 18/24. חלק מהשגיאות נבעו מטעות אחת: המודל העניק שם לא נכון לתכונת מספר סדרתי של מגזין – serial_number במקום serialNumber כנדרש. שגיאות נוספות קרו משום שהמודל לא הגדיר setters לתכונות פרטיות ולא בדק את תקינות הקלט במחלקות המשאבים.

```
class Customer:
    def __init__(self, id: int, name: str, telephone: int):
        self.__id = None
        self.__name = None
        self.__telephone = None

        self.id = id
        self.name = name
        self.telephone = telephone
```

איור 2: קטע קוד שנוצר ע"י ChatGPT ובו רואים כפילות בתכונות שהוגדרו במחלקת הלקוח

מעבר לכך, מרבית הדרישות שפורטו לעיל על עמידה בעקרונות תכנות מונחה עצמים התקיימו כמעט במלואן. הקוד נכתב נקי והשתמש באלגוריתמים ברורים. במחלקת הספריה והלקוח, הקוד בדק את הקלט והחזיר שגיאות במצבים שאינם מציאותיים. הקוד עמד בדרישות ההכמסה והגדיר תכונות פרטיות כמקובל, אך לא הגדיר setters במחלקות המשאבים. כמו כן, המודל הוסיף כנדרש את מחלקת משאב, אך יש לציין כי הוא לא הגדיר אותה כמופשטת אלא כמחלקה רגילה. בנוסף המודל הוסיף מעט קוד מיותר במחלקת הלקוח (ראה איור 2) אך לא חזר על שגיאה זו במחלקות אחרות (אך יש לציין כי כפילות זו אפשרה לבדוק את תקינות הקלט במחלקת הלקוח – שכן במחלקה זו הוגדרה setter כנדרש – פעולה שלא בוצעה במחלקות המשאבים).

לסיכום, ניתן להגיד ש- ChatGPT הקפיד על עקרונות תכנות מונחה עצמים וידע לייצר קוד תקין ואיכותי, העומד במשימות הנדרשות. אך המודל כשל בבדיקת תקינות הקלט (שגיאה בהפשטה) ובהקפדה על מתן שמות מסויימים לתכונות.

Gemini

לעומת זאת, העבודה עם Gemini הייתה יותר מורכבת. ראשית, המודל לא יצר את הקוד בשלמותו והיה צורך לבקש ממנו להשלים חלקים מהקוד. 15 הנחיות נדרשו למודל ע"מ לכתוב את כל הקוד בשלמותו. שנית, גם

```
class Library:
    # ... (שאר הקוד של המחלקה)

    def add(self, item):
        # ... (בדיקות קיום קודם, סוג האובייקט וכו')
        if isinstance(item, Customer):
            self.__customers.append(item)
        elif isinstance(item, Resource):
            self.__resources.append(item)
        else:
            raise TypeError("Invalid item type")
```

איור 3: פונקציה add התומכת בהעמסה שיצר המודל Gemini

למה הפתרון `add` לא מתאים?

הפונקציה `add` ככל הנראה מיועדת להוספת משאבים לספרייה, ולא ללקוחות. לכן, שימוש בה להוספת לקוח לא מתאים מבחינה לוגית ויפיק תוצאות לא צפויות.

פתרון מומלץ:

כדי לתקן את השגיאה, עלינו להגדיר פונקציה חדשה במחלקת `Library` שתטפל בהוספת לקוחות. הפונקציה הזו תקבל אובייקט מסוג `Customer` ותוסיף אותו לרשימת הלקוחות של הספרייה.

דוגמה לקוד:

Python

```python
class Library:
    def __init__(self):
        self.__resources = []
        self.__customers = []
        self.__borrowings = {}  # מילון לשמירת ההשאלות: {customer: [resources]}

    def add(self, item):
        # ... (כמו שהוגדר קודם)

    def add_customer(self, customer):
        """מוסיפה לקוח חדש לספרייה"""
        self.__customers.append(customer)
```

איור 4: Gemini מתעלם מהמתודה שיצר (איור 3) ומבקש ליצור מתודה נוספת להוספת לקוח

כאשר היה הקוד שלם, עדיין נדרשו כמה תיקונים ידניים כדי שהקוד ירוץ כמו שצריך. למשל, המודל הציע ליצור פונקציה בשם add_customer (ראה איור 4) שתטפל בהוספת לקוח (בניגוד לדרישות שבמטלה ולרצון ליצור העמסת מתודות). זאת, על אף שבתחילת השיחה המודל יצר בעצמו פונקצית add תקינה ע"פ דרישות המטלה הכוללת העמסת מתודות (ראה איור 3).

מהרצת הבדיקות על הקוד של Gemini, המודל קיבל ציון מאכזב של 7/24. השגיאות בו היו מגוונות, ביניהן אי הגדרת getters לחלק מהתכונות הפרטיות, אי בדיקת קלט והחזרת שגיאות במקרים שאינם מציאותיים (כגון לקוח שאינו רשום בספריה המעוניין לשאול ספר) ואי עמידה בדרישות המטלה (המודל לא החזיר True במתודה בה הדבר נדרש). יש לציין כי המודל השכיל להוסיף את מחלקת המשאב – אמנם כמחלקה רגילה ולא מופשטת – ואף בדק בחלק מהמקרים קלט ותקינות ערכים (אך לא בכולם).

לסיכום, Gemini אמנם ידע לתת מענה מסויים לדרישות, אך באופן כללי לא עמד כראוי במשימה. למעט עקרון הירושה, המודל לא עמד בעקרונות תכנות מונחה עצמים – לא בדק קלט בחלק מהמקרים, לא הגדיר getters לתכונות פרטיות ולא ידע להשתמש כראוי בהעמסת מתודות. יש לציין כי בבדיקה האם המודל ביצע דריסת מתודות – בתכונות id ו-type – הבדיקה החזירה שגיאה משום שהמודל לא הגדיר כלל, תכונות אלו גם במחלקות היורשות – מה שמהווה כמובן שגיאה בפני עצמה. אך ניתן להחשיב כי המודל השתמש כראוי בדריסת מתודות, משום שבמתודה __str__ כן בוצעה דריסה כנדרש.

השוואה בין המודלים

אם נשווה בין שני המודלים, נראה כי ChatGPT נתן מענה איכותי יותר מעמיתו להנחיה שניתנה. ChatGPT ידע להחזיר שגיאות ברוב המקרים, עמד בעקרון ההכמסה והגדיר תכונות פרטיות כראוי עם getters וsetters, השתמש בצורה טובה בירושה, דריסה והעמסת מתודות.

שני המודלים ידעו לתת מענה בשפה העברית ושניהם הוסיפו מדעתם את מחלקת המשאב כדי לעמוד בעקרון הירושה, אך לא כמחלקה מופשטת. בנוסף, שני המודלים הגדירו כמו שצריך את שמות המחלקות והמתודות, אם כי Gemini נדרש למספר הנחיות רב יותר כדי להשלים את כל הקוד. לעומת זאת, שני המודלים כשלו בבדיקות קלט בעת הגדרת משאבים חדשים.

להלן טבלה המסכמת את השגיאות שעלו בהרצת הבדיקות:

שם מודל	מספר הנחיות כולל	שגיאות בסיסיות	שגיאות בהפשטה	שגיאות בהכמסה	שגיאות בירושה	שגיאות ברב צורתיות	ציון סופי
ChatGPT	1	1/7	3/7	1/5	0/4	1/1	**18/24**
Gemini	15	6/7	7/7	3/5	0/4	1/1	**7/24**

5. מסקנות והשלכות

מתוצאות הניסוי עולה כי מודלי שפה גדולים מסוגלים לתת מענה לבעיות תכנות בשפה העברית ומסוגלים לבנות מערכת של מחלקות בשפת Python. עם זאת, קיימים פערים משמעותיים בין המודלים באיכות המענה שהם נותנים לבעיות תכנות. מהניסוי עלה כי המענה שChatGPT נתן היה מהיר ואיכותי יותר מעמיתו Gemini מכל הבחינות. בקוד של ChatGPT היו פחות שגיאות, הן בפן הסמנטי והן בפן של תכנות מונחה עצמים.

לעומת זאת, עולה מתוצאות הניסוי כי מודלי שפה גדולים עלולים לייצר קוד שלא עומד לגמרי בהנחיות שניתנו. שני המודלים לא השכילו לבדוק את תקינות הקלט בכל המקרים ולא החזירו שגיאות במקומות לא מציאותיים (הגדרנו את זה כשגיאה בהפשטה). במקרים מסויימים, המודל לא סיפק תכונות שנתבקשו (Gemini לא סיפק תכונות id וtype ציבוריות) או סיפק תכונות באיות שונה ממה שנתבקש (ChatGPT סיפק תכונת serial_number במקום serialNumber).

מתוצאות הניסוי שלנו ניתן להשליך על השימוש במודלי שפה גדולים בתחומים שנידונו בראש המאמר. **בפן התעשייתי**, מומלץ למתכנתים לבחור היטב את מודל השפה בו ישתמשו ולבדוק אותו באופן יסודי לפני שיעשו בו שימוש משמעותי. על המתכנתים לבחור במודלים הנותנים מענה מהיר שאינו נופל באיכותו – כמו ChatGPT בניסוי שבוצע. בנוסף, בעת השימוש במודלי שפה גדולים, יש לשים לב לדרישות שלא בוצעו במלואן. לצורך זה, מומלץ לבצע בדיקות יחידה (unit tests) – כדי לוודא שלא נפלו שגיאות בקוד, לבדוק את תקינות הקלט ואת שמות התכונות.

בפן האקדמי, מומלץ למרצים למדעי המחשב לתת לסטודנטים מטלות הכוללות תנאים לא טריוויאליים שיאתגרו את מודלי השפה, כדי לוודא שלא ניתן יהיה להעתיק ישירות קוד שיצר מודל שפה גדול. מומלץ אפילו לבדוק את המענה שנותן מודל שפה פופולארי למטלה עצמה – כפי שבוצע בניסוי. ע"פ מענה זה ניתן להתאים את המטלה כך שתקשה על הסטודנטים לתת למודל לפתור את מטלותיהם.

לדעתי, על המרצים לעודד את הסטודנטים להשתמש במודלי שפה גדולים לצורך פתרון מטלות. עידוד זה, בליווי הנחיות, יגרום לסטודנטים לרכוש כישורים חשובים כמו קריאה ביקורתית, זיהוי שגיאות ותיקונן. בנוסף, שימוש במודלי שפה גדולים ייתן לסטודנטים ניסיון בתחום זה, הנמצא בשימוש גדול בתעשייה.

בנוסף, ניתן אפילו ליצור מטלה ייעודית שתבקש לבצע משימה תכנותית בעזרת מודל שפה גדול (ראה רעיון דומה במאמר מספר 2). ניתן לדרוש מהסטודנטים להשתמש במודל שפה שנותן מענה

פחות איכותי – כמו Gemini בניסוי שבוצע – ולבקש מהם לזהות ולתקן את השגיאות שהמודל מבצע.

לסיום, אזכיר כי הניסוי בוצע בשפה העברית ובגרסה החינמית של המודלים. ייתכן כי שימוש בשפה האנגלית או בגרסה בתשלום יניב תוצאות מעט שונות.

6. סיכום

לסיכום, מודלי שפה גדולים הפכו בשנים האחרונות לכלים משמעותיים בעולם התכנות, המציעים יכולות מרשימות לצד מספר אתגרים. ביצענו ניסוי הבוחן את יכולותיהם של שני מודלים מובילים – ChatGPT ו-Gemini – להתמודד עם משימות תכנות מורכבות, תוך דגש על עמידה בעקרונות התכנות מונחה עצמים – הפשטה, הכמסה, ירושה ורב צורתיות.

תוצאות הניסוי הראו כי מודלי שפה גדולים מסוגלים לתת מענה בשפה העברית וליצור קוד תקין בשפת Python, העומד באופן חלקי בעקרונות תכנות מונחה עצמים. הניסוי הראה כי ChatGPT סיפק מענה מהיר ואיכותי יותר, בעוד Gemini הציג ביצועים פחות מספקים. עם זאת, אף אחד מהמודלים לא עמד באופן מלא בדרישות, בעיקר בתחום בדיקת תקינות הקלט והקפדה על דרישות לא סטנדרטיות.

השימוש במודלי שפה גדולים בתעשייה ובאקדמיה טומן בחובו פוטנציאל רב. בתעשייה, כלים אלו יכולים להאיץ תהליכי פיתוח, אך חשוב לשלבם עם בדיקות יחידה ובקרה אנושית על מנת להבטיח את תקינות הקוד. באקדמיה, יש לעודד שימוש מושכל בכלים אלו, תוך פיתוח מטלות שמאתגרות את המודלים ומחייבות את הסטודנטים לחשיבה ביקורתית ולמידה פעילה.

להערכתי, מודלי שפה גדולים אינם תחליף למתכנתים, אלא כלי עזר חשוב המשלב אוטומציה עם יצירתיות אנושית. עם התפתחות הטכנולוגיה, פוטנציאל הכלים הללו צפוי להתרחב, ולהוביל לשינויים מרחיקי לכת בתעשייה ובאקדמיה. בעזרת שילוב מושכל ונכון, מודלים אלו יכולים לשמש ככלי עזר משמעותי למתכנתים וסטודנטים, לייעל ולקצר תהליכים ולהוות מקור למידה חשוב במקצוע התכנות.

7. ביבליוגרפיה

1. Cipriano, B. P. & Alves, P. (2023). GPT-3 vs Object Oriented Programming Assignments: An Experience Report. *ITiCSE 2023: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 61-67.

2. Cipriano, B. P. & Alves, P. (2024). "ChatGPT Is Here to Help, Not to Replace Anybody" - An Evaluation of Students' Opinions On Integrating ChatGPT In CS Courses. *arXiv: 2404.17443*.

3. Cipriano, B. P. & Alves, P. (2024). LLMs Still Can't Avoid Instanceof: An Investigation Into GPT-3.5 ,GPT-4 and Bard's Capacity to Handle Object-Oriented Programming Assignments. *ICSE-SEET '24: Proceedings of the 46th*

International Conference on Software Engineering: Software Engineering Education and Training, 162-169.

4. Climent, L. & Arbelaez, A. (2023). Automatic assessment of OOP assignments with unit testing in Python and a real case assignment. *Comput. Appl. Eng. Educ* (31), 1321-1338.

5. Wang, S., Ding, L., Shen, L., Luo, Y., Du, B. & Tao, D. (2024). OOP: Object-Oriented Programming Evaluation Benchmark for Large Language Models. *Findings of the Association for Computational Linguistics ACL 2024*, 13619–13639.

6. Zhang, J., Cambronero, J., Gulwani, S., Le, V., Piskac, R., Soares, G. & Verbruggen, G. (2024). PyDex: Repairing Bugs in Introductory Python Assignments using LLMs. *Proceedings of the ACM on Programming Languages, 8*, 1100-1124.