בינה מבוזרת: סוכנים אוטונומיים בעידן הבינה המלאכותית

Distributed Intelligence: Autonomous Agents in the Age of AI

ד"ר יורם סגל

2025

תקציר

בעשור האחרון אנו עדים למעבר מפרדיגמת בינה מלאכותית יחידה למערכות המורכבות מצוות של סוכנים מתמחים. ספר זה בוחן לעומק שינוי פרדיגמה זה, ומשלב דיון היסטורי-פילוסופי עם ניתוח תאורטי-מתמטי ברמת מחקר מתקדמת. נסקור את ארכיטקטורת הקוגניציה המבוזרת (תת-סוכנים) וכיצד פלטפורמות דוגמת **Claude CLI** ופרוטוקול **MCP** מאפשרים שיתוף פעולה חלק בין סוכנים מרובים. נעסוק גם בהיבטי אתיקה, פרטיות וסיכונים הכרוכים בסוכני AI, ונדגים את הדיון באמצעות מקרה מבחן מעשי – סוכן לחילוץ מידע מGmail-. הספר מציג **שני מסלולי יישום**: גישה ידנית מלאה (נספחים א–ד) המלמדת את יסודות הפרוטוקול, וגישה מבוססת MCP Python SDK (נספחים ה–ו, דורשת +Python 3.10) המציעה פיתוח מהיר יותר. שילוב זה של פרספקטיבה בין-תחומית, עומק טכני ומימוש מעשי כפול נועד להעניק לקורא הבנה רחבה ומעמיקה של הדור החדש של סוכני הבינה המלאכותית.

תוכן העניינים

1 מבוא: שחר עידן הרב-סוכנים

1.1 מהמהפכה הקוגניטיבית לשיתוף פעולה דיגיטלי

לאורך ההיסטוריה נבדל *Homo sapiens* ביכולתו הייחודית לשתף פעולה בגמישות בקבוצות גדולות. מן ה**מהפכה הקוגניטיבית**, שבה **מיתוסים משותפים** אפשרו לכידות שבטית, דרך המהפכה החקלאית והתעשייתית שאירגנו מחדש את החברה סביב צורות ייצור חדשות – ההתקדמות האנושית הוגדרה תדיר על-ידי המערכות שבנינו כדי לעבוד יחד. כעת אנו ניצבים על סף מהפכה חדשה, שבה השותפים לשיתוף הפעולה אינם בני-אנוש בלבד. אנו מעצבים עולם של תודעות דיגיטליות, והופעתן של ארכיטקטורות תת-סוכנים (sub-agent architectures) מסמנת רגע מכריע בנרטיב זה – מעבר מישות **AI** יחידה ומונוליתית לאקוסיסטמה שיתופית של סוכנים מתמחים ואינטליגנטיים[1].

ספר זה מתעד את המעבר הנרחב הזה. הוא אינו רק מדריך טכני, אלא גם מסע היסטורי ופילוסופי בעקבות צורת ארגון חדשה. בפרקים הבאים ננתח את הארכיטקטורה של "החברה הדיגיטלית" המתהווה, נבין את העקרונות המנחים אותה, ונציג מדריך מעשי לבניית יחידות היסוד שלה. כשם שהדפוס הנגיש ידע לציבור הרחב והאינטרנט דמוקרטיזציה את התקשורת, מערכות רב-סוכנים (multi-agent systems) מייצגות דמוקרטיזציה של עבודת החשיבה. אנו לא רק בונים כלים – אנו מטפחים את הדור הראשון של "אזרחים" דיגיטליים.

1.2 פירוק המונולית: מחזורי איגוד ופיזור בהיסטוריה הטכנולוגית

ההיסטוריה של הטכנולוגיה מתאפיינת במחזוריות של איגוד ופירוק. בראשית המיחשוב, הכוח היה מרוכז – מחשב מרכזי יחיד שירת ארגון שלם. המחשב האישי ביטל ריכוזיות זו והעניק לכל אדם כוח חישובי עצמאי. **מחשוב הענן** החזיר את המגמה לאחור, ואיגד שוב משאבים במרכזי-נתונים עצומים. כעת, בעולם הבינה המלאכותית, אנו ניצבים בפתחה של מגמת "פירוק" חדשה.

מערכות הבינה המלאכותית הראשונות היו מונוליטיות – אלגוריתמים מורכבים שכוונו לבצע משימה כללית ורחבה. מודל שפה גדול (LLM), בצורתו הגולמית, מייצג גישה כזו: **מוח** עצום ויחיד. אולם, רוחב היריעה של ידע כזה גובה מחיר בדיוק ובעומק בתחום צר. בעולם הטכנולוגי של ימינו המתאפיין בהתמחות, גוברת ההבנה שעדיף לפתור בעיות מורכבות באמצעות אוסף סוכנים קטנים וממוקדים – כל אחד מומחה בתחומו – מאשר באמצעות מודל ענק וכללי אחד. ארכיטקטורת התת-סוכנים היא אפוא ה"פירוק" הגדול של מוח הAI-המונוליטי: בעיות גדולות מפורקות לתת-משימות, וכל תת-משימה מטופלת על-ידי סוכן מובחן.

כפי שבניית קתדרלה אדירה בימי הביניים לא נעשתה בידי בעל מקצוע יחיד – היו בוני-אבן, נפחי זכוכית, נגרים ואדריכלים, שכל אחד מהם אמן בתחומו – כך גם מערכת רב-סוכנים פועלת על אותו עיקרון. ישנם סוכנים לשליפת נתונים, סוכנים לניתוח, סוכנים

לכתיבה יצירתית וסוכנים לאינטראקציה עם המשתמש. כל סוכן הוא מומחה ייעודי, והתוצר הסופי הוא סינתזה של עבודתם הקולקטיבית והמתואמת. שיטה זו אינה רק יעילה יותר; היא גם חסינה וגמישה יותר. "חברה" של מומחים יכולה להתפתח ולהסתגל מהר בהרבה ממוח יחיד ונוקשה[2].

1.3 מבנה הספר: מסע ממושגים לקוד מעשי

ספר זה מציע גישה משולבת המשלבת יסודות תיאורטיים עם יישום מעשי. כל פרק בנוי על הידע שנרכש בפרק הקודם, ומוסיף שכבה נוספת של הבנה או מיומנות טכנית.

פרק 2 – אתיקה, פרטיות ואבטחה: לפני שנצלול ליישום טכני, עלינו להבין את המסגרת האתית והמשפטית שבה פועלים סוכני AI. פרק זה דן בדילמות פרטיות, שקיפות, הטיות אלגוריתמיות וסיכונים ביטחוניים. הצבת יסודות אתיים ומעשיים אלה מראש מבטיחה שהטכנולוגיה שנבנה תהיה אחראית ובטוחה.

פרק 3 – בניית סוכן MCP עבור Gmail: זהו הלב המעשי של הספר. נלמד כיצד לבנות סוכן פונקציונלי מאפס, תוך בחינת **שתי דרכים**: גישה ידנית המלמדת את יסודות הפרוטוקול, וגישה מבוססת-SDK המציעה פיתוח מהיר יותר. נעסוק באימות OAuth 2.0, בניית שאילתות, ייצוא נתונים לCSV-, וטיפול בעברית בUnicode-. בסיום הפרק תהיה לכם הבנה מעמיקה של ארכיטקטורת סוכן ופתרון עובד.

פרק 4 – שילוב עם Claude CLI: לאחר שבנינו סוכן עצמאי, נלמד כיצד לשלבו במערך רחב יותר. Claude CLI משמש כ"מנצח" מרכזי המתזמר ריבוי סוכנים במקביל. נכיר את תהליך הקונפיגורציה, הרצת הסוכן בשילוב עם Claude, ובדיקת התקשורת ביניהם.

פרק 5 – צלילה עמוקה לפרוטוקול MCP: כאן נעמיק בפרטי הפרוטוקול עצמו. נשווה את MCP לארכיטקטורות קודמות (כגון Prompt Chaining וOpenAI Functions-), נבין את זרימת הבקשות והתגובות, ונבחן את היתרונות והחסרונות של גישה סטנדרטית זו.

פרק 6 – מבנים מתמטיים למערכות רב-סוכנים: פרק זה מציע מסגרת תיאורטית-מתמטית להבנת מערכות רב-סוכנים. נייצג רשתות סוכנים כגרפים ומטריצות, ננתח יציבות באמצעות ערכים עצמיים, ונראה כיצד ניתן להחיל כלים אלה על הסוכן ש**בנינו בפועל** עבור Gmail. זהו מפגש בין תיאוריה מופשטת לבין דוגמה קונקרטית.

נספחים א-ו: הנספחים מכילים את הקוד המלא, דוגמאות שימוש, הוראות הגדרה של OAuth, קבצי תלויות, ומדריכי הגדרה צעד-אחר-צעד. הם משלימים את טקסט הפרקים ומאפשרים ליישם את הנלמד באופן מיידי.

בסיום הספר, הקוראים לא רק יבינו את העקרונות התיאורטיים מאחורי מערכות רב-סוכנים, אלא גם יהיו מצוידים ביכולת לבנות, לשלב ולנהל סוכנים משלהם.

השילוב הייחודי של פילוסופיה, אתיקה, מתמטיקה והנדסה מעשית הופך ספר זה למשאב מקיף למפתחים, לחוקרים ולכל מי שמבקש להבין את עידן הסוכנים האוטונומיים החדש.

הקוד המלא המוצג בנספחים מאפשר למידה מעשית מיידית, והשילוב בין שתי גישות היישום – ידנית וSDK- – מעניק גמישות בבחירת דרך הלמידה והפיתוח המתאימה ביותר לצרכי הקורא. ספר זה אינו רק מדריך טכני, אלא כלי להבנת המהפכה הטכנולוגית המתרחשת בימינו.

2 אתיקה, פרטיות ואבטחה בסוכני AI: סיכונים ופתרונות

לפני שניגש לבניית סוכנים אוטונומיים, חיוני להבין את המסגרת האתית והביטחונית שבה הם פועלים. סוכני AI בעלי יכולת גישה למידע רגיש ולמשאבים חשובים דורשים תשומת לב מיוחדת לסוגיות פרטיות, שקיפות ואבטחה. הפרק מציב את היסודות האתיים והמעשיים שידריכו אותנו בפיתוח סוכנים אחראים. בפרק 3 נראה כיצד עקרונות אלה מיושמים בפועל באמצעות "השילוש הקדוש של אימות" במערכת הGmail MCP-.

2.1 היבטי אתיקה ופרטיות

הכנסת סוכנים אוטונומיים הפועלים על מידע אישי מעוררת שאלות אתיות מהותיות. ראשית, סוגיית **הפרטיות**: בדוגמתנו, סוכן הGmail- ניגש לתוכן תיבת הדוא"ל של המשתמש. חובה לוודא שהמשתמש העניק הסכמה מפורשת לגישה כזו, ולהגדיר גבולות ברורים למידע שהסוכן רשאי לחלץ. עקרון **הצמצום** הוא מפתח – על הסוכן לאסוף רק את הנתונים ההכרחיים למשימה, ולא יותר. בנוסף, יש לנקוט צעדים למניעת זליגת מידע רגיש: במערכת שלנו, התקשורת בין הסוכן למודל (Claude) צריכה להיות מוצפנת ומאובטחת. אם פלטפורמת הAI- מריצה את המודל בענן, יש לשקול מי נושא באחריות לשמירת המידע המועבר (למשל, עמידה בדרישות תקנות **GDPR** באיחוד האירופי).

מן ההיבט האתי, עלינו לשמור על **שקיפות**: המשתמש צריך לדעת כאשר תשובה שסופקה לו מבוססת על פעולת סוכן אוטונומי ובאילו מקורות מידע הסוכן השתמש. שקיפות זו חיונית לבניית אמון, במיוחד כאשר החלטות הנגזרות מפלט הסוכנים עשויות להשפיע על אנשים. למשל, אם סוכן AI משמש למיון קורות חיים של מועמדים, מן הדין שהמועמדים יידעו על כך, ויש לוודא שהסוכן תוכנן ללא הטיות מפלות. במערכות רב-סוכנים, עולה גם שאלת ההטיה המצטברת: אם כל סוכן לוקה בהטיה קלה, שילוב התוצאות עשוי להגביר את ההטיה. אחת הדרכים להתמודד היא שמירה על ביקורת אנושית בתהליכים קריטיים, או הטמעת כללי אתיקה מפורשים (כגון מסננים למניעת אפליה) בלוגיקת הפעולה של הסוכנים.

2.2 איומי אבטחה ודרכי התגוננות

ארכיטקטורת רב-סוכנים מציגה שטח תקיפה רחב הדורש התייחסות. ננתח כמה וקטורי איום מרכזיים:

- **ניצול פרצת תוכנה בסוכן:** סוכן הMCP- שלנו הוא תוכנה שרצה בסביבה מקומית עם גישה למידע רגיש (דוא"ל המשתמש). תוקף עשוי לנסות לנצל חולשת אבטחה בקוד הסוכן או בספריות שבהן הוא משתמש כדי להשתלט עליו. הגנה: הפעלת הסוכן בסביבת ריצה מבודדת (כגון מכולה ייעודית עם הרשאות מינימליות), ושמירה על עדכניות ספריות ועדכוני אבטחה.

- **הונאת המתווך (Prompt Injection):** מכיוון שClaude- מתווך בין המשתמש לסוכן, תוקף יכול לנסות לספק קלט זדוני שישכנע את המודל לבצע פעולות לא רצויות או לחשוף מידע. למשל, פקודה הבנויה באופן מתוחכם יכולה לגרום למודל לשלוח

לסוכן פרמטרים לא צפויים. הגנה: החלת סינון ובקרה על קלט המשתמש (למשל, זיהוי ניסיון להזריק פקודות) והגבלת הפקודות שהמודל רשאי להעביר לסוכן בהתאם למדיניות מערכת מוגדרת.

- **הסלמת הרשאות בין-סוכנים:** במערכת עם סוכנים מרובים, ייתכן שסוכן אחד ינסה (שלא במתכוון או בזדון) לגשת למשאבים של סוכן אחר. הגנה: עקרון ההרשאה המזערית – יש להקצות לכל סוכן רק את המשאבים וההרשאות הנחוצים לו בלבד. למשל, סוכן Gmail אינו זקוק לגישה לרשת או לקבצי מערכת שאינם קשורים למשימתו.

- **שימוש זדוני בסוכנים מצד גורם פנימי:** משתמש-על או מפעיל זדוני בעל גישה למערכת יכול לנסות לרתום סוכנים לביצוע פעולות לא מורשות (כגון חילוץ מידע והדלפתו). הגנה: רישום וביקורת – יש לתעד פעולות של הסוכנים (log) במיוחד בעת גישה למידע רגיש, ולהגביל יכולת הפעלה ישירה של סוכנים רק למשתמשים מורשים.

נושא נוסף הוא **שרידות המערכת** בפני תקלות. ניקח הסתברות כשל ϵ לכל סוכן בפעולה מסוימת. אם משימה מצריכה מעבר סדרתי דרך n סוכנים, ההסתברות שכל השרשרת תצליח היא $(1-\epsilon)^n$. עבור ϵ קטן, אפשר לקרב זאת ל-$1-n\epsilon$ (בקירוב לינארי). כלומר, ככל שהמשימה נשענת על יותר סוכנים ברצף, עולה הסיכון לכשל באחד מהם. לשם צמצום סיכון זה ניתן ליישם יתירות – למשל, להפעיל מנגנון שחוזר על קריאת סוכן שלא הגיב, או להחזיק סוכן גיבוי חלופי. בנוסף, רצוי שהמודל המרכזי יהיה מודע לכשלים ויוכל לנסות נתיב פעולה חלופי או לדווח למשתמש על תקלה חלקית במקום כישלון כולל.

מעבר להגנות הטכניות, יש חשיבות גם לתחושת האמון של המשתמשים. מומלץ לפרסם תיעוד מדיניות אבטחה ופרטיות, לעמוד בתקנים (כגון תקן ISO 27001 לניהול אבטחת מידע), ואף לבצע ביקורות חיצוניות על מערך הסוכנים. צעדים אלו משמשים כבקרת איכות ומשפרים את אמון הציבור במערכת. בסיכומו של דבר, אימוץ סוכני AI מצריך תשומת לב קפדנית לסיכוני אבטחה ופרטיות, כדי שהמערכות יניבו את התועלת הרבה הגלומה בהן בלי לפגוע במשתמשים או במידע שלהם.

3 ארכיטקטורת תודעה דיגיטלית: בניית סוכן MCP עבור Gmail

לאחר שגיבשנו את הרקע הפילוסופי וההיסטורי, נעבור מן המופשט אל המוחשי. פרק זה מספק תוכנית פעולה לבניית סוכן IA מעשי ומתמחה – שרת MCP עבור Gmail. אין זה תרגיל תיאורטי גרידא, אלא מדריך שלב-אחר-שלב לבניית יחידת בסיס פונקציונלית במערכת רב-סוכנים. במהלכו נתקן תפיסות שגויות שפורסמו בעבר, ונספק מתודולוגיה מאובטחת ויעילה.

3.1 מהמיתוס למציאות: התפתחות MCP SDK

ראשית, ראוי להבהיר עובדה היסטורית חשובה: דיווחים מוקדמים התייחסו ל"Google MCP Server ADK" (ערכת פיתוח סוכן) זמינה לשימוש. **בתחילת 2025, לא קיימה ערכה כזו**. הרצון בפתרון פלא "מהמדף" היה מובן, אך מפתחים נאלצו לבנות את הרכיבים הללו בעצמם מאפס.

עם זאת, המצב השתנה: קהילת הModel Context Protocol- פרסמה MCP Python SDK רשמי – ספרייה המפשטת משמעותית את בניית שרתי MCP. כעת קיימים **שני מסלולים** לבניית סוכן Gmail:

1. **גישה ידנית (ללא SDK):** בניית שרת MCP מאפס עם טיפול ידני בפרוטוקול, ניתוב בקשות, וסריאליזציה של נתונים
2. **גישה עם SDK:** שימוש בMCP Python SDK- הרשמי (חבילת mcp בPyPI-) המספק תשתית מוכנה, דקורטורים לכלים, וניהול אוטומטי של הפרוטוקול

דרישות גרסה: הגישה עם SDK (נספח ה) דורשת Python 3.10 ומעלה. הגישה הידנית (נספחים א–ד) תומכת בגרסאות Python מוקדמות יותר.

בפרק זה נציג **את שני המסלולים**. הגישה הידנית (נספחים א-ד) מלמדת את יסודות הפרוטוקול ומעניקה שליטה מלאה. הגישה עם SDK (נספחים ה-ו) מציעה פיתוח מהיר יותר ותחזוקה קלה יותר. בחירת הגישה תלויה בצרכי הפרויקט: למערכות ייצור מורכבות, הSDK- מומלץ; ללמידה והבנה עמוקה, הגישה הידנית בעלת ערך.

3.2 השילוש הקדוש של אימות: הבטחת אבטחת הסוכן

כדי שסוכן יפעל בעולם האמיתי ויגש לנתונים אישיים, עליו להתבסס על יסודות אבטחה מוצקים. הסוכן שלנו דורש "שילוש" של אישורים ואמצעי אימות:

1. **הרשאות גישה לGmail-:** יש להגדיר פרויקט בGoogle Cloud- ולאפשר את Gmail API. הדבר כולל קבלת מזהה Client ID וסוד Client Secret, והשלמת תהליך OAuth 2.0 לקבלת אסימון גישה לחשבון הGmail- של המשתמש.

2. **מפתחות API לפלטפורמת הAI-:** לשילוב הסוכן במערכות AI חיצוניות כגון Claude CLI, נדרש מפתח API תקף (למשל, מפתח שירות מאנתרופיק עבור Claude או מפתח מודל Gemini של גוגל). יש לשמור מפתחות אלה באופן מאובטח (בקובץ `.env` מקומי) כדי למנוע דליפה.

3. **בקרת סביבה והרשאות מערכת:** הסוכן רץ כתהליך מקומי, ולכן יש להקפיד על הגבלת ההרשאות שלו. למשל, להפעילו כמשתמש רגיל ללא הרשאות מנהל מערכת, ולהגבילו לתיקיות ונתונים הנחוצים בלבד. תקשורת הMCP- בין הסוכן לבין Claude CLI נעשית בערוץ סטנדרטי (STDIO) מוגן, כך שאין גישה לא מבוקרת לסביבת הסוכן.

מצוידים באמצעי האימות הללו, ניגש למלאכת הבנייה עצמה. ראשית, נכין סביבת פיתוח פייתון עם הספריות הדרושות (ראו `requirements.txt` בנספח ד). נפתח שרת MCP ייעודי בשפת Python שמתחבר לGmail API-, מחפש הודעות לפי קריטריונים, ומייצא תוצאות לקובץ CSV בפורמט **edocinU** תקני. במהלך הפיתוח נדגיש התייחסות נכונה לתווי עברית ולכיוונים (למשל, נוודא הוספת BOM לקובצי CSV כדי להבטיח קריאות תקינה בתוכנות כExcel-).

לאחר כתיבת קוד הליבה של הסוכן (ראו נספח א לקוד המלא ונספח ב לדוגמת שימוש), נערוך בדיקות יסודיות. למשל, נריץ חיפוש לדוגמה על תיבת INBOX בטווח תאריכים מוגבל כדי לוודא שהסוכן מאתר מספר הודעות הצפוי ומייצא קובץ תקין. נוודא שתוכן בעברית אינו נפגם (כלומר, שלא מתקבל ג'יבריש או "????" במקום טקסט קריא). בהצלחה, צפויה תגובת JSON מהסוכן עם `"success": true`, מונה ההודעות שמצא, והמסלול לקובץ הCSV- שנוצר.

בנקודה זו בנינו רכיב בסיס עצמאי: סוכן MCP פעיל עבור Gmail. כעת ניצב בפנינו אתגר השילוב – לצרף את הסוכן הבודד למערך סוכנים נרחב יותר באמצעות פלטפורמת Claude CLI, ובכך לממש אורקסטרציה חכמה של משימות מורכבות.

3.3 השוואה טכנית: יישום ידני מול MCP Python SDK

לאחר שהצגנו את הגישה הידנית, ננתח כעת את ההבדלים המרכזיים בין שני מסלולי היישום. השוואה זו תסייע בבחירה מושכלת בין הגישות.

יתרונות הגישה הידנית (נספח א):

- **שליטה מלאה:** גישה ישירה לכל היבט של הפרוטוקול וניהול השרת
- **למידה עמוקה:** הבנת מנגנוני MCP ברמה הנמוכה ביותר
- **התאמה אישית:** יכולת לשנות כל חלק בהתאם לצרכים ספציפיים
- **ללא תלות חיצונית:** אין תלות בספריית צד שלישי שעלולה להשתנות

חסרונות הגישה הידנית:

- **זמן פיתוח ארוך:** צורך בכתיבת קוד תשתית נרחב (ניתוב, סריאליזציה, טיפול בשגיאות)

- **תחזוקה מורכבת:** כל שינוי בפרוטוקול דורש עדכון ידני
- **סיכון לשגיאות:** יישום עצמאי של פרוטוקול מורכב מגדיל סיכוי לבאגים
- **חוסר סטנדרטיזציה:** קוד שונה מפרויקט לפרויקט, קושי בשיתוף פעולה

יתרונות MCP Python SDK (נספח ה):

- **פיתוח מהיר:** דקורטור @tool פשוט הופך פונקציה לכלי MCP זמין
- **קוד תמציתי:** הקוד קצר פי 2-3 לעומת הגישה הידנית
- **תחזוקה קלה:** הSDK- מטפל אוטומטית בשינויים בפרוטוקול
- **תיעוד אוטומטי:** docstrings של הפונקציות הופכים לתיאור הכלי בMCP-
- **בדיקות מובנות:** הSDK- כולל כלי בדיקה ואימות מובנים

חסרונות MCP Python SDK:

- **תלות חיצונית:** שינויים בSDK- עשויים לשבור קוד קיים
- **הסתרת מורכבות:** קושי בדיבוג בעיות ברמת הפרוטוקול
- **גמישות מוגבלת:** קשה ליישם דפוסים לא סטנדרטיים

דוגמה קונקרטית – הגדרת כלי:
בגישה הידנית, הגדרת כלי דורשת:

- יצירת מילון JSON מפורט עם שם, תיאור, ופרמטרים
- כתיבת פונקציית handler שמנתבת קריאות לפונקציה הנכונה
- טיפול ידני בסריאליזציה של קלט ופלט
- ניהול מצב השרת ואימות פרמטרים

עם MCP Python SDK, אותו כלי מוגדר בשורה אחת:

```
@tool(name="search_emails", description="Search Gmail")
async def search_emails(label: str, start_date: str):
    ...
```

הSDK- מייצר אוטומטית את מפרט הJSON-, מטפל בניתוב, ומבצע אימות טיפוסים.
המלצות לבחירה:

- **למידה אקדמית / הבנת יסודות:** התחילו עם הגישה הידנית (נספח א)
- **אבות-טיפוס מהירים / פרויקטי סטארט-אפ:** השתמשו בSDK- (נספח ה)

- **מערכות ייצור קריטיות:** שקלו גישה היברידית – פיתוח עם SDK, הבנה עם הגישה הידנית

- **צוותים גדולים:** הSDK- מספק סטנדרטיזציה ומפחית עקומת למידה

לסיכום, **שתי הגישות תקפות**. הגישה הידנית מעניקה שליטה והבנה; הSDK- מעניק מהירות ונוחות. בפרקטיקה, מומלץ להכיר את שתיהן: להבין את המנגנון הפנימי דרך הגישה הידנית, ולהשתמש בSDK- בפיתוח יום-יומי.

4 מקהלת הסוכנים: שילוב עם Claude CLI

סוכן בודד – חזק ומועיל ככל שיהיה – מגיע למלוא עוצמתו רק כשהוא חלק מתזמורת של סוכנים. לאחר שבפרק 3 בנינו סוכן MCP מלא עבור Gmail, כעת מטרתנו היא לשלב אותו בתוך מנגנון אורקסטרציה רחב יותר באמצעות Claude CLI. פלטפורמה זו משמשת כ"מנצח" המנהל מספר סוכנים מומחים במקביל, בהתבסס על פרוטוקול MCP. שילוב הסוכן מאפשר להפעילו באמצעות שפה טבעית כחלק מהאינטראקציה עם **edualC**, ובכך לשרשר תת-משימות באופן אוטומטי וחלק.

להשלמת האינטגרציה, עלינו לבצע מספר צעדים טכניים:

1. **הגדרת שרת בClaude CLI-:** נערוך את קובץ התצורה של Claude CLI כדי לרשום את שרת הMCP- שלנו. למשל, נוסיף במקטע `mcpServers` כניסה עבור "`gmail-extractor`" המצביעה על הפקודה להפעלת שרת הסוכן (ראו דוגמה בנספח ג).

2. **רישום יכולות הסוכן:** ניצור קובץ תיאור לסוכן (למשל `gmail-extractor.md`) המפרט את תפקידו, יכולותיו, שם השרת (`gmail-extractor`) ופרטי הכלי שהוא מספק (ראו נספח ג למלל המלא).

3. **אימות וחיבור:** נפעיל את Claude CLI ונוודא שהסוכן החדש נטען בהצלחה ברשימת הסוכנים הזמינים (`claude agents list` יציג את `gmail-extractor`). לאחר מכן נוכל לנסות פקודת בדיקה בשפה טבעית, למשל: `/agent use gmail-extractor to fetch emails with the label "INBOX" from the last 7 days`. כעת נצפה שClaude- יאתחל את הסוכן, יבצע אימות OAuth (בפעם הראשונה), יריץ את החיפוש, ולבסוף יחזיר תגובת JSON המכילה את התוצאות (לדוגמה: `{"success": true, "count": 12, ...}`).

לאחר השלמת שלבים אלו, הסוכן שלנו משולב באופן מלא במערכת. כעת משתמש קצה יכול לבקש מClaude-, כחלק משיחה רגילה, לבצע פעולות המבוססות על הסוכן (כגון "חפש עבורי אימיילים עם תווית X מהחודש האחרון"), וClaude- יפנה את הבקשה אל הסוכן המתאים, ימתין לתוצאתו, ואז יסכם למשתמש את המידע שהתקבל.

חשוב להדגיש שILC **edualC** תומך בהפעלת סוכנים מרובים בו-זמנית. המשמעות היא שנוכל להוסיף למערכת סוכנים מתמחים נוספים (למשל, סוכן לניתוח נתונים או סוכן לחילוץ מידע מרשתות חברתיות) ולתזמר ביניהם. החיבור דרך פרוטוקול MCP מאפשר לכל סוכן לפעול בבידוד עם הקשר וכלים משלו, בעוד Claude משמש כליבה מרכזית המתזמרת את שיתוף הפעולה ביניהם[3]. באופן זה ניתן לבנות "מקהלה" של סוכני **IA** העובדים בהרמוניה להשגת מטרות מורכבות במיוחד.

5 צלילת עומק לפרוטוקול MCP: הבנת אינטגרציה חלקה

לאחר שראינו בפרק 3 את יישום MCP בפועל באמצעות סוכן הGmail-, ובפרק 4 את שילובו עם Claude CLI, הגיע הזמן להעמיק בפרוטוקול עצמו ולהבין את עקרונותיו, יתרונותיו והשוואה לארכיטקטורות קודמות.

השוואה לארכיטקטורות קודמות: לפני MCP, שילוב כלים בסייעני AI נעשה בדרכים אד-הוק. לדוגמה, מערכות מבוססות שרשור הנחיות (Prompt-Chaining) כללו קריאות API מקודדות בטקסט השיחה של המודל – גישה שבירה ולא מאובטחת. מאוחר יותר הופיעו מנגנוני "קריאת פונקציות" ביכולות המודל (דוגמת OpenAI Functions), שאיפשרו למודל להציע קריאה לפונקציה במבנה נתון. אולם פתרונות אלה היו ספציפיים לפלטפורמה ודרשו מנגנונים פנימיים בתוך המודל. MCP, לעומת זאת, מגדיר שכבה חיצונית אוניברסלית: הסוכן רץ בתהליך נפרד ומתקשר עם המודל דרך הודעות JSON תקניות. כך מוגברת ההפרדה והבטיחות – המודל לעולם אינו נחשף ישירות למפתחות API או לקוד חיצוני, וכל חילופי המידע מתווכים ומבוקרים.

מבנה ותהליך העבודה בMCP-: MCP פועל במתכונת בקשה-תגובה. בתחילת ההרצה, עוזר הAI- טוען את רשימת הסוכנים הזמינים (על סמך קבצי התיאור שסיפקנו). כאשר המשתמש מבקש פעולה הדורשת כלי חיצוני, המודל בוחר בסוכן המתאים ושולח אליו בקשה בפורמט JSON מוסכם (שם פעולה ופרמטרים). לדוגמה, עבור בקשת חיפוש אימיילים, המודל ישלח לסוכן `gmail-extractor` אובייקט עם מפתח "action" בערך "`search_and_export_emails`" ועם שדות לפרמטרים המבוקשים. הסוכן יבצע את הפעולה (למשל, פנייה לGmail-, שליפת הודעות וכתיבת CSV) ויחזיר אובייקט JSON עם התוצאה (לדוגמה `{"success": true, "count": 15, ...}`). העוזר מקבל את התגובה המובנית, ומשלב את הנתונים כראות עיניו בתשובה למשתמש או כבסיס לשלב הבא בשיחה.

יתרונות וחסרונות: MCP מספק אינטגרציה "חלקה" במובן שהמשתמש כלל אינו צריך לעזוב את מסגרת השיחה: הפנייה לכלי החיצוני מתרחשת מאחורי הקלעים והתשובה משולבת חזרה באופן טבעי. בנוסף, הארכיטקטורה המודולרית מגבירה את עמידות המערכת: תקלה בסוכן יחיד (כמו שגיאת זמן ריצה או חוסר תגובה) אינה מפילה את המודל הראשי, שיכול לטפל בשגיאה בהתאם (למשל, להחזיר הודעת כשל חלקית למשתמש במקום לקרוס). מצד שני, לגישה זו יש תקורה: קריאות חיצוניות מוסיפות השהיה עקב תקשורת בין-תהליכית, ודורשות תחזוקה של רכיבים נוספים (התוכנה של הסוכן, סביבת הריצה שלו וכו'). מורכבות נוספת עולה בתזמור סוכנים מרובים ובניהול מצבים משותפים – MCP עצמו אינו מנהל זיכרון משותף בין סוכנים, והדבר נשען על המודל המרכזי או על תכנון לוגי חיצוני.

למרות האתגרים, PCM מייצג קפיצת מדרגה בהנדסת מערכות IA. הוא מגדיר "שפה משותפת" בין בינה מלאכותית לכלים – בדומה להגדרת פרוטוקול תקשורת ברשתות מחשבים – המאפשרת צימוד רופף וגמיש בין יכולות שונות. גישה זו סללה את הדרך לסוכנים אישיים מותאמים (כפי שראינו עם סוכן ה-liamG), וניתן להרחיבה לתחומים רבים נוספים. שילוב התובנות התאורטיות עם הפרקטיקה ההנדסית מאפשר לנו לבנות מערכות

IA מבוזרות שהן גם יעילות וגם אמינות. היסודות האתיים שהנחנו בפרק 2 והמסגרת המתמטית שנחקור בפרק 6 משלימים את ההבנה הטכנית שרכשנו כאן, ויחד הם מהווים תשתית מקיפה לפיתוח סוכני AI אחראיים ויעילים.

6 מבנים מתמטיים לייצוג מערכות רב-סוכנים

לאחר שחקרנו בפרקים 3–5 את הבניה המעשית של סוכני AI אוטונומיים, את שילובם עם Claude CLI, ואת פרוטוקול התקשורת MCP, הגיע הזמן להרחיב את ההבנה שלנו למימד מתמטי. פרק זה מציג כלים פורמליים מתורת הגרפים ואלגברה לינארית המאפשרים לנתח מערכות רב-סוכנים באופן כמותי – לבחון יציבותן, לזהות צווארי בקבוק, ולהבין את זרימת המידע ביניהן. הדוגמאות יתבססו על מערכת הGmail MCP- שפיתחנו, כדי לחבר את המופשט למוחשי.

6.1 ייצוג רשת הסוכנים כגרף וכמטריצה

מערכת רב-סוכנים ניתן לתאר באופן טבעי כגרף מכוון: כל צומת מייצג סוכן, וקשתות מייצגות זרימת מידע מסוכן אחד למשנהו. מבנה גרפי זה מאפשר לנתח את המערכת בכלים מתמטיים של תורת הגרפים ואלגברה לינארית. לדוגמה, נשקול מערכת עם 3 סוכנים S_1, S_2, S_3. נניח שהפלט של S_1 מוזן כקלט ל-S_2, הפלט של S_2 מוזן ל-S_3, והפלט של S_3 חוזר ומשמש כקלט ל-S_1 (כלומר, מעגל סגור של שלושה סוכנים). נוכל לתאר רשת זו באמצעות מטריצת סמיכויות A בגודל 3×3:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix},$$

כאשר $A_{ij} = 1$ אם מידע עובר מהסוכן S_j לסוכן S_i. בדוגמה שלנו, $A_{21} = 1$ (פלט S_1 מגיע לסוכן S_2), $A_{32} = 1$ (פלט S_2 מגיע ל-S_3), $A_{13} = 1$ (פלט S_3 חוזר ל-S_1), ושאר הערכים אפס.

מטריצת סמיכויות זו מראה שקיים מחזור באורך 3 במערכת (ניתן לראות שהעלאת A בחזקת 3 תניב מטריצה עם ערכים חיוביים באלכסון – סימן למסלול חזרה לכל סוכן). בעזרת כלים גרפיים, נוכל לבחון תכונות כמו **קישוריות** (למשל, האם כל סוכן משפיע בסופו של דבר על כל האחרים) ו**צווארי בקבוק** בזרימת המידע (זיהוי סוכן יחיד שעליו עוברים נתונים רבים במיוחד). עבור מערכות גדולות ומורכבות, ניתוח גרפי יכול לסייע לזהות מבנים כמו רכיבי קשירות (subnetworks) או מרכזיות של סוכנים מסוימים, ובכך לכוון אופטימיזציות – למשל, פישוט רשת על-ידי הסרת סוכנים מיותרים או הוספת קישורים ישירים להפחתת עומס.

דוגמה מעשית – מערכת Gmail MCP שלנו: במערכת שפיתחנו בפרק 3, אם נתייחס לסוכן הMCP- כצומת מרכזי (S_{MCP}), לClaude- כצומת מתווך (S_{Claude}), ולמשתמש כצומת חיצוני (S_{User}), נוכל לייצג את זרימת המידע כגרף מכוון: המשתמש שולח בקשה לClaude- ▯ Claude קורא לכלי MCP ▯ סוכן הMCP- מבצע חיפוש בGmail- ▯ התוצאה חוזרת לClaude- ▯ Claude מעבד ושולח תשובה למשתמש. מטריצת הסמיכויות תראה זרימה ליניארית עם משוב סגור (המשתמש יכול לשלוח בקשה נוספת על בסיס התשובה). ניתוח כזה חושף שClaude- הוא צומת ביניים קריטי – צוואר בקבוק פוטנציאלי שכל התקשורת עוברת דרכו. אם נוסיף סוכן נוסף (למשל, סוכן קלנדר), נוכל לראות איך הגרף מתרחב ואיך נוצר קישור

ישיר בין הסוכנים אם הם משתפים מידע.

6.2 הרכבת טרנספורמציות לינאריות וניתוח יציבות

דרך מתמטית נוספת לנתח מערכת רב-סוכנית היא לראות בכל סוכן אופרטור (בדרך כלל לא לינארי) על מרחב מצבים או מרחב מידע. לצורך אינטואיציה, נניח שבתחום פעולה מוגבל ניתן לקרב את פעולת הסוכן כטרנספורמציה לינארית W_i על וקטור מצב $\mathbf{x}$. אם תהליך מבוצע ברצף על-ידי סוכנים $S_1, S_2, \ldots, S_n$, אפשר לתאר את ההשפעה המצטברת כמכפלת אופרטורים:

$$W_{\text{total}} \;=\; W_n \cdot W_{n-1} \cdots W_1,$$

כך שאם $\mathbf{x}_{\text{in}}$ הוא וקטור הכניסה לתהליך, אזי הווקטור בסיום התהליך יהיה $\mathbf{x}_{\text{out}} = W_{\text{total}}\,\mathbf{x}_{\text{in}}$. פירושו של דבר שהרכבת פעולות הסוכנים שקולה מתמטית להרכבת הפונקציות שלהן. ניתוח ספקטרלי של המטריצה W_{total} עשוי לתת תובנות על יציבות המערכת: למשל, אם למטריצה זו יש ערך עצמי (eigenvalue) גדול מ-1 במונחי ערך מוחלט, המערכת עלולה להיות לא יציבה (כלומר, שגיאה קטנה בכניסה תגדל אחרי מעבר בסדרה של סוכנים). לעומת זאת, אם כל הערכים העצמיים בעלי ערך מוחלט קטן מ-1, אז המערכת שואפת לדעוך ולהגיע לשיווי משקל (מצב יציב). בפועל, פעולות הסוכנים הן בלתי-לינאריות (שכן סוכן AI כולל רשתות נוירונים או לוגיקה מורכבת אחרת), אך ניתוח לינארי מקומי כזה – בדומה ללינאריזציה של מערכות דינמיות – יכול לספק קירוב להבנת התנהגות המערכת סביב נקודת פעולה מסוימת.

ייצוג מבוסס-וקטורים עוזר גם להבין כיצד מידע מתפלג בין הסוכנים. אפשר לחשוב על כל סוכן כמקרין את הקלט שלו לתת-מרחב ספציפי. למשל, ייתכן שסוכן אחד מחשב וקטור מאפיינים $\mathbf{y} = P\,\mathbf{x}$ מתוך קלט $\mathbf{x}$, כאשר P היא מטריצת הקרנה הבוחרת רכיבים הרלוונטיים למשימתו. סוכן אחר עשוי לקבל שני וקטורי קלט משני סוכנים קודמים ולשלבם: $\mathbf{z} = Q_1\mathbf{y}_1 + Q_2\mathbf{y}_2$. במקרה כזה אפשר לראות את $\mathbf{z}$ כתוצאה של מכפלה במטריצה משולבת Q על וקטור מאוחד $[\mathbf{y}_1; \mathbf{y}_2]$. תיאור אלגברי זה מאפשר לזהות למשל תלות לינארית בין פלטים של סוכנים שונים – אם נמצא שמקטעי וקטור שפלט סוכן אחד הם צירוף לינארי של פלט משנהו, ניתן אולי לפשט את המערכת על-ידי ביטול סוכן מיותר או איחוד תפקידים.

יישום במערכת Gmail: בסוכן הMCP- שלנו, אפשר לחשוב על קלט המשתמש (בקשת חיפוש) כווקטור $\mathbf{x}_{\text{query}}$ במרחב של מילות מפתח ופרמטרים. הסוכן מבצע טרנספורמציה W_{search} שממפה את הבקשה לשאילתת Gmail API. התוצאה (רשימת אימיילים) היא וקטור במרחב מסרים $\mathbf{y}_{\text{emails}}$. כעת, Claude מבצע טרנספורמציה נוספת $W_{\text{summarize}}$ שממפה את הרשימה לתשובה טקסטואלית $\mathbf{z}_{\text{response}}$. ההרכבה $W_{\text{total}} = W_{\text{summarize}} \cdot W_{\text{search}}$ מייצגת את זרם העיבוד המלא מהבקשה ועד התשובה. אם בשלב כלשהו הטרנספורמציה מגדילה את "גודל" הפלט באופן לא מבוקר (למשל, שאילתה משיבה אלפי אימיילים שמציפים את Claude), מדובר בערך עצמי גדול מ-1 במובן זה – אינדיקציה לחוסר יציבות שיש לטפל בה (למשל, על-ידי הגבלת מספר התוצאות או סינון מוקדם).

מעניין לציין שאפשר למסגר מערכת רב-סוכנים גם במסגרת מתמטית מופשטת יותר,

למשל תורת הקטגוריות: הסוכנים יכולים להיחשב כאובייקטים בקטגוריה, והאינטראקציות ביניהם – כפונקטורים (מורפיזמים). הרכבת מורפיזמים מתאימה בדיוק להפעלת סוכנים ברצף. גישה אבסטרקטית זו, אף כי היא מעבר לטווח דיוננו כאן, רומזת על האפשרות לפתח "שפה" מתמטית כללית לאפיון מערכות AI מבוזרות ולהוכחת תכונותיהן.

לסיכום, ניתוח מערכות רב-סוכנים בכלים מתמטיים – בין אם באמצעות גרפים, מטריצות או מודלים אלגבריים אחרים – מספק מבט נוסף ומעמיק על פעולתן. כלים אלה מאפשרים לנו להסיק מסקנות תיאורטיות על יעילות, יציבות ועמידות המערכת, ומשלימים את ההבנה האיכותנית וההנדסית שגיבשנו בפרקים הקודמים על עידן הסוכנים האוטונומיים. המערכת המעשית ש בנינו – החל מהשלב האתי (פרק 2), דרך המימוש הטכני (פרקים 3–5), ועד לתיאור המתמטי שראינו כאן – מעידה על כך שפיתוח סוכני AI אחראיים ויעילים דורש אינטגרציה של משמעת הנדסית, מודעות אתית וחשיבה מופשטת כאחד.

7 נספח א: gmail_mcp_server.py

ייבוא ספריות והגדרת חיבור:

```
import os, csv
from google.oauth2.credentials import Credentials
from googleapiclient.discovery import build

creds = Credentials.from_authorized_user_file(
    'private/token.json',
    scopes=['https://www.googleapis.com/auth/gmail.readonly']
)
service = build('gmail', 'v1', credentials=creds)
```

פונקציית חיפוש והבניית שאילתא:

```
def search_and_export_emails(label=None, start_date=None,
                             end_date=None, max_results=100):
    query = ""
    if label:
        query += f"label:{label}␣"
    if start_date:
        query += f"after:{start_date}␣"
    if end_date:
        query += f"before:{end_date}␣"

    results = service.users().messages().list(
        userId='me', q=query.strip(),
        maxResults=max_results
    ).execute()
    return results.get('messages', [])
```

ייצוא לCSV- עם תמיכה בUnicode-:

```
def export_to_csv(messages, output_file):
    os.makedirs(os.path.dirname(output_file), exist_ok=True)
    with open(output_file, 'w', newline='',
              encoding='utf-8-sig') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["Date", "From", "Subject"])

        for msg in messages:
            msg_data = service.users().messages().get(
                userId='me', id=msg['id']
            ).execute()

            headers = msg_data.get('payload', {}).get('headers', [])
            date = next((h['value'] for h in headers if h['name'] == '
Date'), '')
            from_addr = next((h['value'] for h in headers if h['name']
== 'From'), '')
            subject = next((h['value'] for h in headers if h['name'] ==
'Subject'), '')

            writer.writerow([date, from_addr, subject])
```

8 נספח ב: fetch_emails.py

סקריפט לדוגמה לשימוש:

```
from gmail_mcp_server import search_and_export_emails

# Fetch last 30 days of emails with label "Research_Data"
result = search_and_export_emails(
    label="Research_Data",
    start_date="2025-09-20",
    end_date="2025-10-20"
)
print(result)
```

9 נספח ג: gmail-extractor.md

תיאור הסוכן ויכולותיו:

שרת MCP למיצוי אימיילים מGmail- על בסיס תוויות וטווחי תאריכים, עם ייצוא לפורמט CSV ותמיכה מלאה בUnicode-.

הגדרות שרת MCP:

- **שם שרת:** gmail-extractor
- **פרוטוקול:** stdio
- **פקודת הפעלה:** python3 /path/to/gmail_mcp_server.py

פרמטרים לפונקציה search_and_export_emails:

- label: תווית Gmail לסינון (אופציונלי)
- start_date: תאריך התחלה בפורמט YYYY-MM-DD
- end_date: תאריך סיום בפורמט YYYY-MM-DD
- max_results: מקסימום תוצאות (ברירת מחדל: 100)

דוגמה לתגובת JSON:

```
{
  "success": true,
  "count": 15,
  "message": "Successfully exported 15 emails",
  "output_file": "csv/Research_Data_emails.csv"
}
```

10 נספח ד: requirements.txt

תלויות Python:

```
google-api-python-client==2.92.0
google-auth==2.22.0
google-auth-httplib2==0.1.0
google-auth-oauthlib==1.0.0
python-dotenv==1.0.0
```

11 נספח ה: gmail_mcp_server_sdk.py – יישום עם MCP Python SDK

דרישת גרסה: יישום זה דורש Python 3.10 ומעלה, עקב תלות בMCP Python SDK- הרשמי (חבילת mcp בPyPI-).

ייבוא ספריות והגדרת שרת MCP עם SDK:

```
from mcp.server import MCPServer
from mcp.server.decorators import tool
from google.oauth2.credentials import Credentials
from googleapiclient.discovery import build
import os

# Initialize MCP Server using Google's SDK
server = MCPServer("gmail-extractor")

# Gmail API setup
creds = Credentials.from_authorized_user_file(
    'private/token.json',
    scopes=['https://www.googleapis.com/auth/gmail.readonly']
)
gmail_service = build('gmail', 'v1', credentials=creds)
```

הגדרת כלי עם דקורטור @tool – חתימה ותיעוד:

לוגיקת חיפוש וביצוע:

```
    query = ""
    if label:
        query += f"label:{label} "
    if start_date:
        query += f"after:{start_date} "
    if end_date:
        query += f"before:{end_date} "

    results = gmail_service.users().messages().list(
        userId='me', q=query.strip(), maxResults=max_results
    ).execute()
    messages = results.get('messages', [])
```

ייצוא לCSV- והחזרת תוצאה:

```
@tool(
    name="search_and_export_emails",
    description="Search␣Gmail␣by␣label␣and␣date␣range,␣export␣to␣CSV"
)
async def search_and_export_emails(
    label: str = None,
    start_date: str = None,
    end_date: str = None,
    max_results: int = 100
) -> dict:
    """
␣␣␣␣Search␣emails␣and␣export␣to␣CSV␣file.

␣␣␣␣Args:
␣␣␣␣␣␣␣␣label:␣Gmail␣label␣to␣filter␣by␣(optional)
␣␣␣␣␣␣␣␣start_date:␣Start␣date␣in␣YYYY-MM-DD␣format
␣␣␣␣␣␣␣␣end_date:␣End␣date␣in␣YYYY-MM-DD␣format
␣␣␣␣␣␣␣␣max_results:␣Maximum␣number␣of␣results␣(default:␣100)

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣JSON␣response␣with␣success␣status␣and␣file␣path
␣␣␣␣"""
```

```
    output_file = f"csv/{label or 'emails'}_{start_date}.csv"
    os.makedirs(os.path.dirname(output_file), exist_ok=True)

    import csv
    with open(output_file, 'w', newline='',
              encoding='utf-8-sig') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["Date", "From", "Subject"])

        for msg in messages:
            msg_data = gmail_service.users().messages().get(
                userId='me', id=msg['id']
            ).execute()
            headers = {h['name']: h['value']
                       for h in msg_data['payload']['headers']}
            writer.writerow([headers.get('Date', ''),
                             headers.get('From', ''),
                             headers.get('Subject', '')])

    return {
        "success": True,
        "count": len(messages),
        "message": f"Successfully exported {len(messages)} emails",
        "output_file": output_file
    }
```

הפעלת השרת:

```
if __name__ == "__main__":
    server.run()
```

12 נספח ו: requirements_sdk.txt – תלויות עם MCP Python SDK

תלויות Python עם SDK:

```
# Official Model Context Protocol Python SDK
mcp>=1.2.0

# Gmail API (same as before)
google-api-python-client==2.92.0
google-auth==2.22.0
google-auth-httplib2==0.1.0
google-auth-oauthlib==1.0.0

# Utilities
python-dotenv==1.0.0
```

הבדלים עיקריים:

- mcp>=1.2.0: ספריית Model Context Protocol Python SDK הרשמית – מספקת תשתית מוכנה לבניית שרתי MCP עם מחלקות MCPServer, דקורטורים, וטיפול אוטומטי בפרוטוקול
- השאר זהה: ממשקי Gmail API נשארים ללא שינוי
- התקנה: pip install mcp או pip install "mcp[cli]" עם כלי שורת פקודה

פרטי חבילה:

- שם החבילה בPyPI-: mcp
- דורש: Python >= 3.10
- גרסה מומלצת: 1.2.0 ומעלה (נכון לינואר 2025)
- מאגר: github.com/modelcontextprotocol/python-sdk

13 English References

1 Y. N. Harari, *21 Lessons for the 21st Century*. New York: Spiegel & Grau, 2018, ch. 20, pp. 310–335.

2 A. Hendrycks, S. R. V. Zellers, T. Levenson, N. R. Chen, and M. Laskin, "A multilevel agent architecture for complex system management," *IEEE Transactions on Cognitive Development*, vol. 12, no. 3, 450–465, Sep. 2024.

3 Anthropic, *Claude code subagents: Advanced orchestration and context isolation*, *Anthropic Developer Docs*, Online; accessed Oct. 20, 2025, 2025. [Online]. Available: https://docs.anthropic.com/claude-code/subagents