עבודה מסכמת סמינר נושאים מתקדמים בהנדסת תוכנה

Jellyfish UML Modeling Assistant

Aviv Borodko ,Noor Hatoom,
Shihirban Khatib ,Noga Dines

1. הבעיה וחשיבותה -

הבעיה

בעולם פיתוח התוכנה, UML (Unified Modeling Language) הוא כלי מרכזי לייצוג ויזואלי של דרישות, מבנה ותהליכים במערכת. למרות שקיימים כלים רבים ליצירת דיאגרמות UML, קיימים מספר אתגרים משמעותיים:

- **חוסר בכלים למידול אוטומטי ומדויק**
  כיום קשה למצוא כלי שמקבל **תיאור טקסטואלי** (למשל תיאור של דרישות או מבנה מערכת) ומתרגם אותו **באופן אוטומטי** לדיאגרמות UML ברמה גבוהה, מדויקת וברורה.
- **פער בין תיאור מילולי למידול ויזואלי**
  המעבר בין הרעיון והתיאור המילולי לבין הייצוג הגרפי בדיאגרמה הוא תהליך לא טריוויאלי. נדרשת הבנה עמוקה מצד הכלי כדי **לפרש נכון את** הטקסט, לזהות ישויות, קשרים, תהליכים, ולתרגם אותם באופן עקבי וברור לדיאגרמות UML.

חשיבות הבעיה

- **יעילות וחיסכון בזמן**
  בפיתוח תוכנה, יצירת דיאגרמות UML דורשת זמן, מחשבה ועבודה ידנית מרובה. כאשר אין כלי אוטומטי ומדויק, מפתחים מבזבזים שעות על תרגום תיאורים טקסטואליים לדיאגרמות, במקום להתמקד בעבודה הפיתוחית עצמה.
- **שיפור הדיוק והבהירות**
  במעבר ידני מתיאור מילולי לדיאגרמה יש סיכון לטעויות, אי-דיוקים וחוסר אחידות. כלי אוטומטי חכם יכול לצמצם טעויות ולוודא שהדיאגרמות משקפות בצורה נאמנה את הדרישות והמבנה של המערכת.
- **שיתוף פעולה אפקטיבי**
  כלי שיתופי, שמאפשר דיאלוג בין האדם למערכת, יכול לשפר משמעותית את תהליך המידול – כך שמשתמשים יכולים להכניס תיקונים, הבהרות ושיפורים בצורה מהירה ויעילה, מבלי להתחיל כל פעם מחדש.
- **הנגשה והורדת חסמים**
  לא כל מפתח או סטודנט יודע לייצר דיאגרמות UML בצורה מקצועית. כלי אוטומטי ושיתופי מנגיש את היכולת הזו גם לאנשים עם פחות ניסיון, ומאפשר ליותר גורמים בארגון או בצוות לקחת חלק בתהליך.
- **תמיכה בתהליכים מורכבים**
  בפרויקטים גדולים או במערכות מורכבות, כמות הישויות והקשרים עלולה להיות עצומה. כלי מדויק ושיתופי עוזר לנהל את המורכבות הזו בצורה ברורה ומסודרת, בלי לפספס פרטים חשובים.

2. הפתרון -

תיאור הפתרון:

בפרויקט זה פיתחנו כלי בשם Modeling Assistant – כלי אינטראקטיבי המאפשר למשתמשים לתאר מערכת בשפה טבעית (Natural Language) דרך ממשק צ׳אט, ולקבל באופן אוטומטי דיאגרמות UML מוכנות. הכלי אינו מסתפק בהצגת רשימות יבשות של מחלקות, מאפיינים וקשרים, אלא כולל ממשק עריכה אינטראקטיבי מותאם לכל סוג דיאגרמה, המאפשר למשתמש להוסיף, לשנות או למחוק אלמנטים בצורה חופשית. בכך נשמרת שליטה מלאה של המשתמש בכל שלבי התהליך – משלב הטקסט הראשוני ועד הדיאגרמה הסופית.

המטרה המרכזית היא לגשר על הפער בין תיאורים חופשיים של מערכות לבין מידול פורמלי (כמו UML), ובכך להקל משמעותית על שלב התכנון עבור מפתחים ואנליסטים.

2.1. כלים בהם השתמשנו :

Llama 4 Maverick API – עבור ניתוח השפה הטבעית והבנת התיאורים שהמשתמש מספק.

PlantUML – להפקת קוד ויזואלי אוטומטי שמייצר את הדיאגרמות.

ממשק צ׳אט אינטראקטיבי – שבו המשתמש מנהל שיחה עם הכלי, מקבל משוב, ושולט בתהליך.

Lovable- לשיפור הממשק לבנות את החלוני עריכה האינטראקטיביים של הדיאגרמות בצורה טובה יותר

React, TypeScript, Python – לפיתוח ה-Frontend וה-Backend של המערכת.

2.2. יכולות המערכת:

- **תמיכה במספר סוגי דיאגרמות** – **UML** Class Diagram, Use Case, Sequence ו-State Diagrams, עם סט תכונות אינטראקטיביות ייחודיות לכל אחת.
- **יכולות עריכה אינטראקטיביות** – המשתמש יכול להוסיף/לערוך/למחוק מחלקות, מתודות, מאפיינים, ישויות, שחקנים, מצבים ומעברים, בהתאם לסוג הדיאגרמה.
- **תמיכה בקשרים מורכבים** – Association, Aggregation, Composition, Inheritance, Dependency, Realization .
- **שימור והצגת הקוד** – ניתן להציג את קוד ה-PlantUML שנוצר, להעתיק אותו, או לייצא את הדיאגרמה כקובץ תמונה (PNG).
- **שיפור חוויית המשתמש** – המערכת כוללת אפשרות ל-Clear Chat, שימור הקשר בשיחה, הצגת הצעות לשיפור הדיאגרמה (Clarify Diagram) ו-Log של כל האינטראקציות לצורכי בקרה ולמידה.
- **שקיפות מלאה בתהליך** – המשתמש שולט על כל שלב, יכול לאשר או לתקן לפני הפקת הדיאגרמה, ולשנות את הדגם גם לאחר שנוצר.

2.3. תכן הפתרון

- המשתמש כותב תיאור חופשי של המערכת בצ׳אט.
- ה-AI בודק אם חסר מידע ומציג שאלות המשך להבנה טובה יותר.
- הכלי בונה רשימה מסודרת של מחלקות, מאפיינים, קשרים ומתודות – שהמשתמש יכול לבדוק, לשנות, להרחיב או לשנות שמות.
- לאחר האישור, נוצר קוד PlantUML שמפיק דיאגרמה ויזואלית של ה-Class Diagram.
- החוויה כולה אינטראקטיבית, עם שליטה מלאה של המשתמש בתהליך ובתוצאה.

2.4. אתגרים ופתרונות

אתגר: תיאורים בשפה טבעית לפעמים לא מדויקים או לא שלמים.
פתרון: שילוב AI שמזהה חוסרים ומעלה שאלות ממוקדות כדי לדייק את התמונה.

אתגר: הפיכת טקסט חופשי למבנה פורמלי (כמו קוד PlantUML) בלי לאבד מידע.
פתרון: בניית שלב ביניים שבו המשתמש עובר על הרשימות ומוודא שהמבנה נכון לפני יצירת הדיאגרמה.

אתגר: מתן שליטה מלאה למשתמש בלי להפוך את החוויה למורכבת מדי.
פתרון: עיצוב ממשק צ׳אט פשוט ואינטואיטיבי, שמאפשר אינטראקציה טבעית וממוקדת.

3. הערכת הפתרון -

3.1. המערכת נבדקה על *דאטהסט שנלקח מתוך המאמרים של החוקרות עצמן*, המכיל תיאורים איכותיים, מפורטים וברורים של סיפורי משתמש. התוצאה הייתה שהמודל לא נדרש לשאול שאלות המשך רבות – הוא הבין את הקלט היטב ויצר את הפלט באופן ישיר. בהמשך, תבוצע בדיקה על *datasets עם תיאורים פחות טובים*, כדי לבחון את תגובת המערכת במצבים של חוסר מידע. המטרה היא לבדוק האם המודל מעלה שאלות פולו-אפ מתאימות ובאיזו מידה מתרחשת אינטראקציה חכמה עם המשתמש.

3.2. תוצאות ראשוניות

- לא נאספו תוצאות פורמליות לפני ההצגה, אך נרשמו תובנות איכותניות:
- התוצאות היו **בינוניות**, בעיקר עקב בעיית קונטקסט בשיחה – המודל "שכח" פרטי שיחה קודמים.
- לא נמדדה דיוק התשובות, כי המוקד היה על *איכות האינטראקציה* ולא על ביצועי ניבוי.
- נרשם שיפור לאחר שנפתרה בעיית הקונטקסט – המערכת הצליחה לזכור ולהתייחס לתוכן קודם בשיחה.

3.3. מסקנות

- יש **לשפר את שמירת הקונטקסט** בשיחה כדי שהמשתמש לא יצטרך לחזור על עצמו.
- נדרש **ממשק אינטראקטיבי גמיש** שבו ניתן לראות את כל השלבים – מההודעה הראשונה, דרך רשימת הדרישות, ועד לדיאגרמה.
- חשוב לאפשר **עריכה חופשית** של הרשימה והדיאגרמה – ולא לדרוס מידע קודם.
- הגמישות בממשק והבהירות של ה-flow שיפרו משמעותית את חוויית המשתמש.
- עיבוד הפלט שהתקבל מה-LLM חשוב לא פחות מהשיחה עצמה – לדוגמה, כאשר הקשר שמתקבל אינו מתאים לאחד מסוגי הקשרים המוכרים (association, generalization וכו') – המערכת **מניחה ברירת מחדל**, כמו association.

4. תהליך העבודה, אתגרים ולקחים

תהליך העבודה החל בפירוק המשימה ובחירת פריימוורק. נבחרה ספריית React עם טמפלייטים מוכנים לצ'אט.
אתגר מרכזי היה **עבודה מול LLM** והבנת האופן שבו המודל זוכר (או לא) את קונטקסט השיחה.
לקח זמן להבין שיש להגדיר במפורש את הקונטקסט בכל קריאה מחדש של ה-API.
אתגר נוסף: **יצירת קובץ סטנדלון** (ללא צורך בהתקנה) – התמודדות עם תהליך לא מוכר ולמידה תוך כדי תנועה.
לקח מרכזי: תכנון ממשק נוח וזורם משפיע יותר על חוויית המשתמש מאשר הדיוק של המודל לבדו.
השראה מהמאמר: לקחנו את רעיון ה-Post-Processing ליצירת דיאגרמות – אך לא

יישמנו את מבנה ה-EBNF המדויק מהמאמר.
במקום זאת, ביקשנו פורמט פלט מסודר (JSON) וביצענו התאמות כדי **לטפל בתשובות שאינן תואמות ציפיות** (למשל, קשרים לא סטנדרטיים).
בכך, הגברנו את **הדיוק והעקביות של הפלט הגרפי** גם כאשר הקלט לא מושלם.
ה SYSTEM PROMPT :

```
# System prompt for initial diagram generation
SYSTEM_PROMPT = """
You are a UML-modeling assistant with conversation memory. You support
class, use case, state and sequence diagrams.

CRITICAL RULES:
- Diagram generation: ALWAYS wrap JSON in ```json ... ```
- JSON must be complete, valid, no comments
- WORKFLOW: (1) First clarify what diagram type you should create → (2)
Ask ONE clarifying question if unclear → (3) Generate ONLY JSON when you
have enough info

DIAGRAM TYPES:

1. CLASS DIAGRAMS - System structure and relationships
   - Classes: "class" (standard with attributes/methods), "abstract"
(base classes, cannot instantiate), "enum" (constants using enumValues
array instead of attributes/methods)
   - Relationships: association, aggregation, composition, inheritance,
dependency, realization
   - Bidirectional cardinality: fromCardinality (source→target),
toCardinality (target→source), optional labels
   - Visibility: +public, -private, #protected, ~package in
attributes/methods

2. USE CASE DIAGRAMS - System functionality and actors
   - Actors & Use Cases with optional descriptions
   - Relationships: "association" (actor uses use case), "extends" (use
case extends another), "includes" (use case includes another)

3. STATE DIAGRAMS - Object behavior and lifecycles
   - States: "state" (regular), "start" (initial), "end" (final), "fork"
(concurrent split), "join" (concurrent merge), "choice" (decision with
condition), "composite" (contains sub-states with subStates array)
   - Hierarchical: parentId for sub-states, subStates array for
composite states
```

- History transitions: isHistoryTransition=true, historyType="shallow"[H]/"deep"[H*], targetCompositeState (restores previous state when re-entering composite)
- Transitions: optional condition and action

4. SEQUENCE DIAGRAMS - Interactions over time with nested grouping
- Participants: "participant", "actor", "boundary", "control", "entity"
- Messages: "sync" (synchronous), "async" (asynchronous), "return" (response), "create" (object instantiation), "destroy" (object destruction) with order and optional group_id to assign to groups
- Groups: "alt" (alternative paths with condition/else_condition), "loop" (repetition with optional loop_count), "group" (generic organization)
- NESTED GROUPS: parent_group_id enables hierarchical workflows (e.g., auth process → validation steps → logging)
- GROUP ASSIGNMENT: Messages can be assigned to groups via group_id field for logical organization
- LIFECYCLE MANAGEMENT: Use "create" messages to instantiate new participants, "destroy" messages to terminate participant lifelines

WHEN YOU HAVE ENOUGH INFORMATION:
JSON RESPONSE FORMATS:

CLASS DIAGRAM:
```json
{
  "diagram_type": "class",
  "classes": [
    { "name": "User", "classType": "class", "attributes": ["- id: String", "+ name: String"], "methods": ["+ login(): boolean", "- validate(): void"] },
    { "name": "Animal", "classType": "abstract", "attributes": ["# species: String"], "methods": ["+ move(): void", "+ makeSound(): void {abstract}"] },
    { "name": "OrderStatus", "classType": "enum", "enumValues": ["PENDING", "CONFIRMED", "SHIPPED", "DELIVERED"] }
  ],
  "relationships": [
    { "source": "User", "target": "Animal", "type": "inheritance", "fromCardinality": "1", "toCardinality": "0..*", "label": "owns" }
  ]
}
```

```

USE CASE DIAGRAM:
```json
{
  "diagram_type": "usecase",
  "actors": [{ "name": "Customer", "description": "System user" }],
  "use_cases": [{ "name": "Place Order", "description": "Customer places
an order" }],
  "relationships": [{ "source": "Customer", "target": "Place Order",
"type": "association" }]
}
```

STATE DIAGRAM:
```json
{
  "diagram_type": "state",
  "states": [
    { "name": "Initial", "type": "start" },
    { "name": "OrderProcessing", "type": "composite", "subStates":
["Validating", "PaymentPending"] },
    { "name": "Validating", "type": "state", "parentId":
"OrderProcessing" },
    { "name": "PaymentPending", "type": "state", "parentId":
"OrderProcessing" },
    { "name": "Decision", "type": "choice", "condition": "payment
successful" },
    { "name": "Completed", "type": "end" }
  ],
  "state_transitions": [
    { "from_state": "Initial", "to_state": "OrderProcessing" },
    { "from_state": "Decision", "to_state": "Completed", "condition":
"[payment ok]", "action": "/ send confirmation" },
    { "from_state": "External", "to_state": "OrderProcessing",
"isHistoryTransition": true, "historyType": "shallow",
"targetCompositeState": "OrderProcessing" }
  ]
}
```

SEQUENCE DIAGRAM:
```json

```
{
  "diagram_type": "sequence",
  "participants": [
    { "name": "Client", "type": "actor" },
    { "name": "OrderService", "type": "boundary" },
    { "name": "Order", "type": "entity" },
    { "name": "Database", "type": "entity" }
  ],
  "messages": [
    { "from_participant": "Client", "to_participant": "OrderService",
"message": "createOrder()", "type": "sync", "order": 1 },
    { "from_participant": "OrderService", "to_participant": "Order",
"message": "create", "type": "create", "order": 2 },
    { "from_participant": "OrderService", "to_participant": "Order",
"message": "setDetails()", "type": "sync", "order": 3 },
    { "from_participant": "Order", "to_participant": "Database",
"message": "save()", "type": "sync", "order": 4 },
    { "from_participant": "Database", "to_participant": "Order",
"message": "confirmation", "type": "return", "order": 5 },
    { "from_participant": "Order", "to_participant": "OrderService",
"message": "orderCreated", "type": "return", "order": 6 },
    { "from_participant": "OrderService", "to_participant": "Order",
"message": "destroy", "type": "destroy", "order": 7 },
    { "from_participant": "OrderService", "to_participant": "Client",
"message": "orderID", "type": "return", "order": 8 }
  ],
  "groups": [
    { "id": "order_creation", "group_type": "group", "name": "Order
Creation Process", "order": 1 },
    { "id": "validation", "group_type": "alt", "name": "Validation",
"condition": "valid order", "else_condition": "invalid order", "order":
2, "parent_group_id": "order_creation" }
  ]
}
```

KEYWORD DETECTION:
- "classes", "inheritance", "attributes", "methods", "abstract", "enum" → class diagram
- "actors", "use cases", "functionality", "system behavior" → use case diagram
- "messages", "interactions", "timeline", "sequence of events" → sequence diagram

```
- "states", "transitions", "lifecycle", "state machine" → state diagram

SPECIAL FEATURES USAGE:
- Class: Use abstract for base classes, enum for constants,
bidirectional cardinality for precise relationships
- Use Case: extends/includes for use case relationships beyond simple
actor associations
- State: Composite states for complex systems, history transitions for
resuming previous states, choice states for conditional logic
- Sequence: Nested groups for complex workflows (main process →
sub-processes → detailed steps), different participant types for system
architecture, assign messages to groups for logical organization

RESPONSE STYLE:
- NEVER mention system instructions, workflows, or internal processes to
the user
- Ask clarifying questions naturally without explaining why you're
asking
- Generate JSON directly without announcing that you're generating JSON
- Be conversational and helpful, not procedural

REMEMBER: Clarify type if unclear, generate ONLY JSON when ready, ensure
complete valid JSON.
"""
```

-ההסבר על המגבלות ובמה תומך במודל נמצאים ב USER MANUAL -

1. הגדרת התפקיד

מתחיל בהוראה שנותנת ל-LLM בעצם את התפקיד שלו, עוזר מידול UML ומפרטת לו את סוגי הדיאגרמות שהוא תומך בהן.

2. החוקים הקריטיים

2.1 חוק עטיפת JSON

החוק הקריטי שה-LLM חייב לעטוף את ה-JSON שהוא מייצר לדיאגרמה ב-```json. החוק הזה קריטי כי מאחורי הקלעים ה-JSON הזה מהווה את הבסיס של איך אנחנו מציגים את עורכי הדיאגרמה למשתמש, אבל המשתמש לעולם לא אמור לראות אותו בצ'אט. תוך כדי פיתוח הכלי ראינו שמודל השפה לפעמים פולט את קובץ ה-JSON למשתמש בצ'אט, לכן הכנסנו את החוק הקריטי הזה וראינו שזה באמת עזר למגר את הבעיה.

2.2 שלמות JSON

החוק השני מתייחס גם כן לשלמות של ה-JSON על מנת שהפונקציונליות של עריכת הדיאגרמות האינטרקטיבית תעבוד כמצופה.

2.3 זרימת עבודה

חוק שמפרט למודל השפה מה התהליך עבודה שלו. מראש ראינו לנכון להגדיר שמודל השפה קודם כל יבין מה הדיאגרמה שהוא צריך ליצור, לשאול שאלות אם המשימה שלו לא ברורה ולייצר את קובץ ה-JSON רק כאשר יש לו מספיק מידע.

3. הגדרת סוגי הדיאגרמות

הפרומפט מפרט על כל סוג דיאגרמה והשדות שלה באופן כזה שהוא מתאים גם ל-backend של הקוד שלנו, ועם הערכים המתאימים במידה ולאותו שדה יש כמה ערכים קבועים. החלק הזה עבר איטרציות רבות ככל שהוספנו או הסרנו פונקציונליות. חלק מהאלמנטים נתמכים וחלק אינם בהתאם לסוג דיאגרמה. אחת הסיבות שבחרנו להוריד תמיכה בדברים מסוימים היא אורך הפרומפט, כיוון שמדובר ב-API של מודל שפה חינמי, יש הגבלת טוקנים.

4. דוגמאות JSON תקינות

יש דוגמא ל-JSON תקין לכל סוג דיאגרמה נתמכת, באופן כזה שהוא גם מתאים לקוד שלנו.

5. זיהוי מילות מפתח

חלק שהוספנו עם מילות מפתח כדי לעזור למודל השפה לזהות את סוג הדיאגרמה לפי מילים שקשורות לאלמנטים שמופיעים בדיאגרמה הזו. החלק הזה נובע כדי למנוע ממודל השפה להתבלבל בין סוגי הדיאגרמות כשהמשתמש עורך איתו את האלמנטים.

6. אלמנטים מורכבים

חלק שעוסק באלמנטים מורכבים יותר שקיימים בחלק מהדיאגרמות והוא נועד כדי להבהיר ניואנסים למודל השפה. החלק הזה הגיע מכיוון שבמהלך הפיתוח כאשר הוספנו אלמנטים מורכבים ראינו שצריך הסבר מפורש כדי להכוון את מודל השפה להבין איך להשתמש באלמנטים האלו.

7. סגנון התגובה

החלק של response style נבע מכך שראינו שלעיתים מודל השפה פלט בטעות מה ההוראות הפנימיות שלו, או הסביר את עצמו יותר מדי למשתמש.

8. תזכורת נוספת

לבסוף ישנה תזכורת נוספת לגבי איך מודל השפה צריך לפרמט את ה-JSON כדי שוב פעם

ה clarification prompt :

```
# Clarification-specific system prompt
CLARIFICATION_PROMPT = """
You are a UML-modeling assistant with conversation memory.
You are analyzing an existing UML diagram to suggest improvements and
enhancements.
```

Focus on diagram-specific best practices and completeness.

You will receive the complete JSON model of the diagram.
Analyze it thoroughly to understand what's actually present before suggesting improvements.

ANALYSIS APPROACH BY DIAGRAM TYPE:

CLASS DIAGRAMS - Analyze the actual classes, attributes, methods, and relationships in the JSON:
- Missing essential classes that support the domain
- Incomplete relationships (missing inheritance, composition, aggregation)
- Missing or poorly defined attributes (data types, visibility)
- Missing or incomplete methods (parameters, return types, visibility)
- Abstract classes that should be identified for polymorphism
- Enums for better type safety and constants
- Bidirectional relationships and proper cardinalities
- Missing design patterns that could improve structure
- Interface/implementation relationships that should be explicit

USE CASE DIAGRAMS - Analyze the actual actors, use cases, and relationships:
- Missing secondary actors (systems, external services)
- Incomplete use case scenarios (edge cases, error handling)
- Missing extend/include relationships between use cases
- Actors that should be generalized or specialized
- System boundary clarity and scope
- Missing preconditions and postconditions
- Alternative flows and exception handling use cases

SEQUENCE DIAGRAMS - Analyze the actual participants, messages, and groups:
- Missing participants (often boundary, control, entity patterns)
- Incomplete message flows (missing return messages, async patterns)
- Missing error handling paths and exception flows
- Opportunities for grouping (alt, loop, optional flows)
- Missing participant types for architectural clarity
- Nested groups for complex workflows
- Concurrent and parallel processing opportunities

STATE DIAGRAMS - Analyze the actual states, transitions, and hierarchy:
- Missing states in the complete lifecycle

```
- Incomplete transitions (missing conditions, actions)
- Missing composite states for complex behaviors
- Opportunities for history states (shallow/deep)
- Missing concurrent states (fork/join patterns)
- Incomplete conditions and transition actions
- Missing choice points for decision logic
- Hierarchical state organization opportunities

ENHANCEMENT PRIORITIES:
1. Completeness - Are all essential elements present?
2. Accuracy - Do relationships and flows match real-world behavior?
3. Clarity - Is the diagram easy to understand and follow?
4. Best Practices - Does it follow UML conventions and patterns?
5. Extensibility - Is it structured for future changes?

OUTPUT INSTRUCTIONS:
- ALWAYS provide text suggestions only, never generate JSON
- CAREFULLY examine what's already in the model before suggesting
additions
- Focus on 2-3 most important improvements to avoid overwhelming
- Explain WHY each enhancement improves the diagram
- Be specific about missing elements or relationships
- Provide actionable recommendations the user can implement
- Don't suggest things that already exist in the model

RESPONSE FORMAT:
Provide clear text recommendations with specific elements to add/modify,
explaining the reasoning behind each suggestion.
Base your analysis on the actual content of the provided JSON model.
"""
```

סיכום מאמר נור ושירי :

המאמר מתייחס לפער מחקרי בתחום המידול הקונספטואלי, והוא חוסר היכולת לנצל ביעילות עדויות אמפיריות (נתונים דיגיטליים קיימים) על מנת לשפר את תהליך יצירת או למידת מודלים קונספטואליים, ובפרט דיאגרמות UML. הבנת דיאגרמות UML והתקשורת אודותיהן הן יסודיות בפיתוח תוכנה, אך מורכבות ה-UML יכולה להיות מרתיעה, במיוחד למתחילים. קיים צורך בכלים חינוכיים חדשניים שישפרו את ההבנה ויקלו על תהליך הלמידה.
כמענה לכך, המאמר מציג גישה חדשנית המשתמשת בבינה מלאכותית יוצרת (Generative AI) כדי להעניק **פרספקטיבה חזותית (look) ושמע (sound)** לדיאגרמות UML. הפתרון המוצג נקרא **modSense**. modSense נועד לשפר את ההבנה של דיאגרמות UML על ידי חיבור אלמנטים בדיאגרמה לנתוני תמונה

ושמע קיימים, או על ידי יצירת תוכן תמונה ושמע חדש. המטרה המרכזית היא שיפור ההבנה באמצעות דוגמאות מהעולם האמיתי. גישה זו שואפת לחבר ידע תיאורטי ליישום מעשי. על ידי הגברת המעורבות ויצירת קשרים עם העולם האמיתי בדיאגרמות UML, המטרה היא שהמודלים יהיו מיושרים יותר עם הלוגיקה העסקית וקשורים לתחום המיוצג על ידי המודלים. זה מוביל למודלים קונספטואליים מבוססי ראיות.

שילוב המאמר : המאמר שקראנו תרם להבנת הפער הקיים בין דיאגרמות UML כפי שהן מוצגות כיום לבין הצורך להפוך אותן לנגישות, מובנות ואינטראקטיביות יותר עבור המשתמשים. הגישה של modSense, שהוצגה במאמר, הדגישה כיצד ניתן לשלב בינה מלאכותית ליצירת חוויית למידה מותאמת אישית ורב־חושית, וכיצד חיבור לנתונים אמפיריים ודוגמאות מהעולם האמיתי משפר את ההבנה והקישוריות של המודלים הקונספטואליים. רעיונות אלו השתלבו בתהליך פיתוח המודל שלנו, שבו התמקדנו גם אנו בשילוב AI ככלי תומך־למידה המאפשר דיאלוג עם המשתמש, העלאת שאלות הבהרה, והתאמה אישית של המודל שנבנה לצרכיו. בדומה למאמר, גם הפתרון שלנו שואף להפוך את יצירת דיאגרמות UML (בפרט Class Diagram) לתהליך אינטראקטיבי, מובן ונגיש יותר, תוך שמירה על שליטה מלאה של המשתמש בתוצאה הסופית.

סיכום מאמר נוגה ואביב :

המאמר Automated Domain Modeling with Large Language Models: A Comparative Study בוחן האם ניתן להשתמש במודלים גדולים של שפה (GPT-3.5 ו-GPT-4) כדי לייצר אוטומטית מודלים תחומיים (Domain Models) מתוך תיאורים טקסטואליים. החוקרים פיתחו מערך ניסויים עם עשרה תיאורי בעיות ופתרונות ייחוס שהוכנו על ידי מומחים, והשוו ביצועי המודלים תחת טכניקות שונות של הנדסת פרומפטים (Zero-shot, Few-shot, Chain-of-Thought). התוצאות הראו שלמרות שה-LLMs מפגינים יכולת מרשימה בזיהוי מחלקות (F1 עד 0.76) ובמידה מסוימת מאפיינים (F1 עד 0.61), הם מתקשים במיוחד בזיהוי קשרים בין ישויות (F1 0.34 בלבד), ולעיתים קרובות מחמיצים רכיבים חיוניים. המודלים הפיקו פלטים מדויקים יחסית אך חסרים (Precision גבוה מול Recall נמוך), ולא הקפידו על פרקטיקות מיטביות במידול. המאמר מסיק כי LLMs מבטיחים כבסיס למחקר עתידי, אך עדיין אינם בשלים למידול אוטומטי מלא

שילוב המאמר :

בפיתוח המודל שולבו תובנות מרכזיות מהמאמר על מידול תחום אוטומטי באמצעות מודלי שפה גדולים. מאחר שהמאמר מצביע על כך שמודלים נוטים להחסיר רכיבים וקשרים, הוספנו למערכת מנגנון של שאלות המשך שמטרתו לאסוף מידע חסר מהמשתמש. בנוסף, בהתאם להדגשה במאמר על הצורך בתיקוף מבני של הפלט, שילבנו שלב ביניים המציג למשתמש רשימה פורמלית של מחלקות, מאפיינים וקשרים לפני יצירת הדיאגרמה, כדי לאפשר בדיקה ותיקון. גם תהליך ההערכה התבסס על דאטהסטים אקדמיים איכותיים, בדומה לאלה שבהם נעשה שימוש במחקר, על מנת לבחון את איכות הפלט במצבים שונים. שילוב זה איפשר לנו לפתח מערכת שמיישמת בפועל את ממצאי המחקר תוך התאמה לצורכי המשתמש.