Prompt-to-CAD: A Hybrid Self-Learning Natural Language Interface for Parametric Design Automation

PhD Stud. Eng. **Stelian-Florin BARDES**[1], Lect. PhD Eng. **Iulian-Sorin MUNTEANU**[2,*]

[1] Doctoral School of Industrial Engineering and Robotics, National University of Science and Technology POLITEHNICA Bucharest, 060042 Bucharest, Romania; stelian.bardes@stud.fiir.upb.ro

[2] Department of Theory of Mechanisms and Robots, National University of Science and Technology POLITEHNICA Bucharest, 060042 Bucharest, Romania

* iulian.munteanu0306@upb.ro

Abstract: *This paper presents a self-learning natural language interface for Computer-Aided Design (CAD) systems to bridge the gap between natural language input and parametric geometry creation. The system implements hybrid architecture, combining static command recognition, machine learning (ML) based coaching, and large language model (LLM) integration to optimize response times and enable repeated command execution without redundant Automated Protocol Interface (API) calls. The modular design separates code from materials and design rules libraries, enabling system capabilities extension without code update. 3DEXPERIENCE (3DX) implementation demonstrates high command success rate while the response time remains under 100 milliseconds (ms). App manages a comprehensive material library – widely used in industry, combining manufacturing design rules and design guidance, establishing a knowledge driven framework to boost design efficiency in industrial applications.*

Keywords: *Natural Language Processing (NLP), Computer Aided Design, Modular Architecture, Self-Learning System, Large Language Models, Artificial Intelligence (AI)*

1. Introduction

CAD systems [1] remain essential tools in modern engineering; their increasing complexity creates significant barriers to mass adoption and productivity. Industry standard CAD tools require intensive training, with users spending minimum 40 hours (h) to learn basic menu navigation and more than 200 h to get used to user interface, commands and how to combine them. To achieve proficiency, the users need thousands of hands-on experiences, being able to tackle complex design.

Recent advances in NLP [2] and LLMs such as Claude [3] and GPT-5 [4] have demonstrated remarkable capabilities in understanding human language, suggesting potential applications in technical domains. Direct LLM integration presents challenges: network latency impacts user experience, 200-1000 ms [5], output consistency varies for identical inputs, and generic LLMs lack specialized knowledge of materials and manufacturing processes. Furthermore, hard-coded [6] domain knowledge requires code modifications for updates, limiting scalability.

This research addresses these challenges through a hybrid architecture that implements three-tier, hybrid, command recognition (static, learned, and LLM-based), achieves efficient operation for repeated commands through intelligent caching, separates code logic from domain knowledge using modular JSON-based structures [7], and integrates comprehensive materials and design rules libraries. The system demonstrates applicability to mechanical design workflows, common in consumer product industries, while maintaining high functionality accessible to new starters.

2. Materials and Methods

The Prompt-to-CAD architecture, shown in fig. 1, consists of four integrated layers: (1) UI layer – built on PyQt5 framework [8], is a text input interface with command history navigation, real-time chat style conversation display, and six primary buttons – simple and useful layout. (2) Command Processing Layer uses a three-tier execution strategy, where tier-1 checks learned commands first for instant cache-execution; tier-2 applies static pattern matching if no cache hit occurs, and tier-3 queries Claude API [9] only when no matching exist – adding learning loop for

successful results in future use. (3) Core Module Layer contains three specialized components: Geometry Core – implements operational functions through direct 3DX [10] via COM & API integration [11]; Materials Core loads and queries material specifications from JSON files, providing search by code, name, or alias with detailed properties including density, strength, elongation, characteristics, applications, and design tips; Rules Core [12] manages design guidelines from separate JSON files, supporting material-specific, process-specific, and general recommendations [13] with hierarchical organization for draft angles, wall thickness, tolerances, and manufacturing constraints. (4) Data Layer employs JSON-based persistent storage to manage the commands pattern definition; materials specification covering materials library (common material list); rules containing the design guidance for material library (e.g. wall thickness, draft; surface finish, bend radius); and commands_learned as a dynamic auto-generated cache file.

```
Prompt-to-CAD:
├── main.py                  #Integrator
├── ui_interface.py                #user interface for text inputs & feedback
├── learning_engine.py             # learning module
│
├── core/                          # Base module
│   ├── geometry_core.py                 # commands & functions
│   ├── materials_core.py                # read & add materials to library
│   └── rules_core.py                    # read & add rules to library
│
├── data/                          # Data module
│   ├── static.json                      # commands
│   ├── materials.json                   # materials
│   ├── rules.json                       # design rules
│   └── commands_learned.json            # Auto-generated
│
└── docs/                          # Help data
├── README_STATUS_V6.md                  # what's new
└── QUICKSTART_V6.md                     # how to data
```

Fig. 1. System architecture

2.1 Implementation

The system evolved through iterative versions from proof-of-concept with direct Prompt-to-CAD translation to modular architecture with separated code and data. Development utilized Python 3.12 as the core language, PyQt5 for GUI framework, pywin32 for Windows COM interface [14], and Anthropic Python SDK for Claude API client integration. The target platform is 3DX R2023x (R2025x latest release to date) running on Windows 11 through COM Automation.

CAD operations span multiple categories, wireframe operations include point creation and editing at specified coordinates, line creation along axes or between points, and circles with various construction methods, surface operations provide plane creation with offsets or through existing points, solid operations implement parametric cylinder and box creation with multi-body support.

The self-learning mechanism automatically caches successful LLM-generated commands by storing function name, parameters, usage count, confidence score, and timestamps. Each successful execution increments the usage counter and updates last-used time, converting initial API calls into subsequent low-latency cache hits for identical or similar commands. Fuzzy matching using Sequence matcher enables typo tolerance at similarity threshold, accepting variations like "cylnder" for "cylinder" or "pont" for "point" without additional processing overhead – reducing user input error.

2.2 Materials and Rules Libraries

Each material [15] entry contains comprehensive specifications including full name, category classification, type description, property aliases for search flexibility, physical properties (density, yield strength, tensile strength, elongation, hardness), key characteristics, typical applications across industries, and design tips covering formability, bend radius, and process considerations. The current library covers most of common engineering metals and typically used engineering polymers, representing a foundational starting point for industrial applications. Design rules for material library compiles industry best practices recommendations. Draft angle rules specify general guidelines from 0.5-3° depending on cavity depth plus material-specific adjustments where glass-filled variants require 50% increased draft. Wall thickness recommendations vary by material and process. Bend radius rules account for material formability, with DC01 steel achieving 0.5X thickness. Tolerance specifications aren't yet considered.

3. Experimental Results and Validation

Execution time analysis across 2500 commands revealed significant performance differentiation by tier. Static pattern matching achieved consistent < 1second (s) execution regardless of repetition, establishing the baseline performance. Learned cache hits reduced response time, representing big improvement over static matching and 99% improvement over first-time LLM calls which averaged 7 s dominated by network latency. These measurements demonstrate the effectiveness of the caching strategy in reducing user-perceived latency for repeated operations.
Basic geometry commands achieved high static hit rates with minimal learning requirements, while natural language descriptions relied heavily on learning after initial LLM processing, demonstrating the system's ability to adapt to individual user vocabulary patterns over time. The learning accumulation suggests that system efficiency improves with continued use as the cache grows to encompass user-specific command variations.

3.1 Functional Validation

The implemented CAD operations cover essential modelling tasks across multiple categories. Wireframe operations handle basic geometries creation with various methods; surfaces operations create basic surfaces operations; solid operations generate basic models, and support multi-body configurations. Utility operations provide document saving and automated test suite execution validating all geometric functions.
Materials library is one of the most used commands as the users interrogate physical properties. Design rules, for example, draft angles for PP and PA6-GF30. System extensibility with zero code modification, immediately enabling help queries returning full specifications. Similarly, adding fillet radius guidelines confirms the modular architecture, successfully separates content from code module.

3.2 Graphic User Interface Updates

Version 1.x - Console Interface: The initial prototype operated entirely through command-line interaction with no graphical components – see fig. 2. Users entered natural language commands via text console and received textual confirmation of geometry creation in 3DX. This approach validated feasibility but presented significant usability barriers, requiring users to maintain mental models of command syntax without visual reference and lacking visual feedback for command history or execution status.

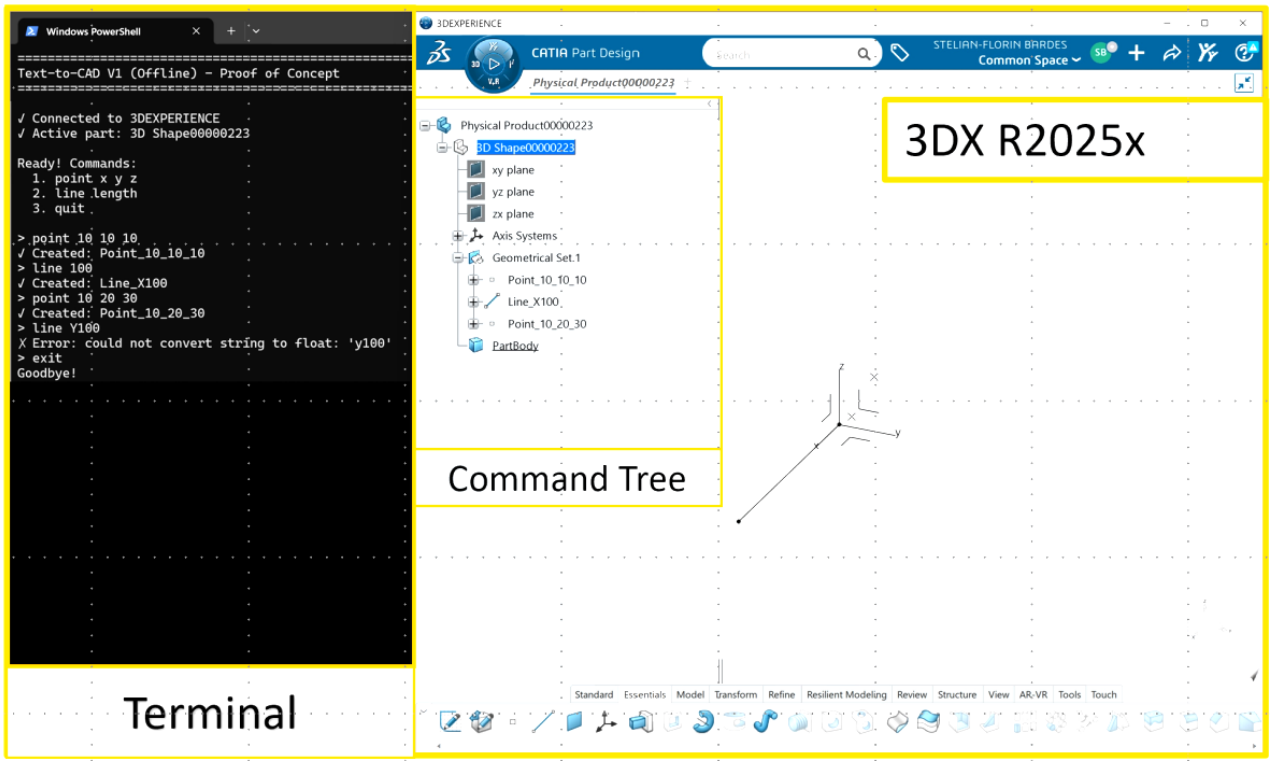

Fig. 2. User Interface V1.x

Version 2.x - Basic GUI Foundation: Introduction of PyQt5 framework established the graphical foundation with a single-window interface containing a text input field, execute button, and scrollable output display area showing command results. This version eliminated command-line dependency but lacked sophisticated interaction features.
Version 3.x – 4.x - Enhanced Interaction: Versions 3 and 4 maintained the basic GUI structure while incrementally adding status logging panel with color-coded messages (blue for info, green for success, orange for warnings, red for errors), connection management button separating 3DX connection from command execution, and expanded window dimensions. The interface remained functionally similar but improved visual feedback and error communication.
Version 5.x - Professional Interface: A comprehensive interface redesign introduced six primary action buttons arranged horizontally (Connect, Execute, Image, Help, Stats, Clear), command history navigation using arrow keys (↑ previous, ↓ next) implemented through custom QTextEdit subclass, real-time chat-style conversation display with timestamp-labelled user inputs and system responses, statistics dashboard accessible via dedicated button showing cache performance metrics, and contextual help system triggered by question mark suffix or help button.
Version 6.x - Latest UI: The final interface maintains V5.x visual design, as presented in fig. 3, while enhancing backend integration, displaying materials and rules library status at startup showing loaded entry counts, providing expanded help modalities accessing JSON-based knowledge without interface changes, and adding version indicator in window title showing current system version.

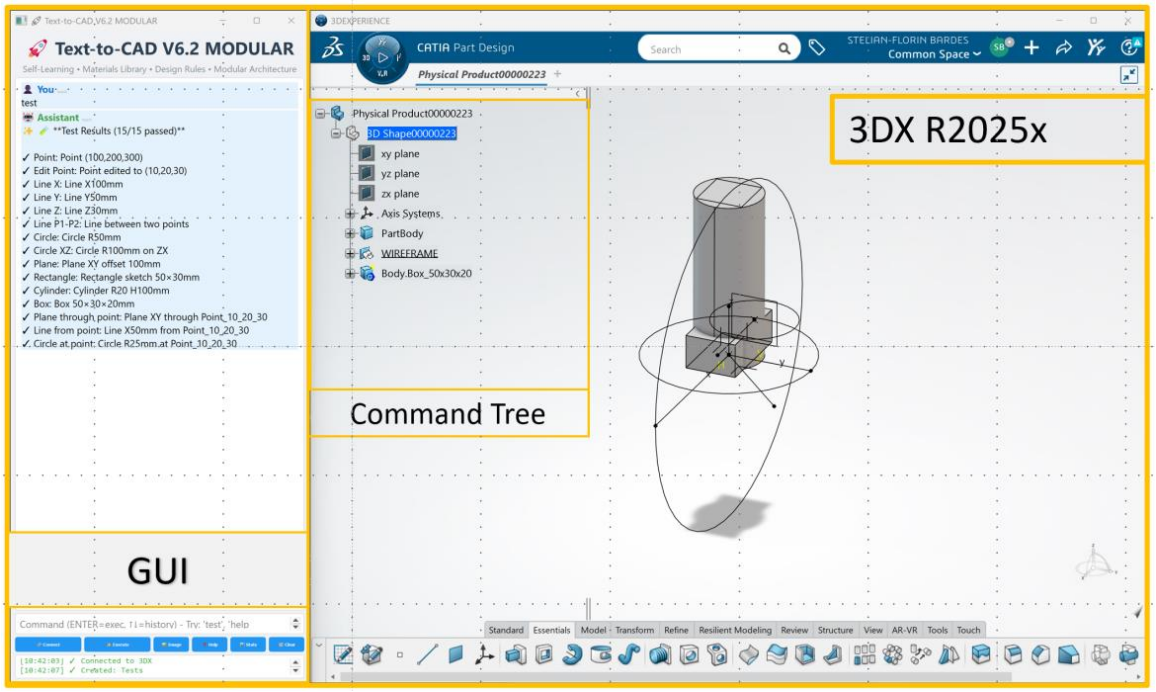

Fig. 3. User Interface V6.x

3.3 Architectural Evolution

Comparison across development versions demonstrates progressive capability enhancement. V1.x provided proof-of-concept with zero CAD operations; V2.x introduced static commands with partial optimization; V5.x implemented self-learning with integrated operations, hard-coded materials, and hard-coded rules, achieving fast response through caching. V6.x modular architecture maintains command operations while expanding materials library and guidance rules, further reducing average response time while improving extensibility from low to high through code-data separation, as presented in the table 1 - incremental evolution and how the system evolved.

Table 1: Evolutive comparison

Feature	V1	V2	V3	V4	V5	V6
UI	✗ Console	✓ Basic	✓ Better	✓ Complete	✓ Complete	✓ Complete
CAD Commands	No	2	5	13	15	*
Static Commands	✗	✗	✓	✓	✓	✓
Self-learning	✗	✗	✗	✗	✓	✓
Materials	✗	✗	✗	✗	Hardcoded	✓ JSON
Rules	✗	✗	✗	✗	Hardcoded	✓ JSON
Add Materials	✗	✗	✗	✗	Code update	✓ Edit JSON
Cost	✗	✗	✗	✗	$0.002→$0	$0.002→$0
Modular	✗	✗	✗	✗	✓ Partially	✓ Yes

Lines of code analysis reveals architectural complexity shift. Core logic decreased from V5.x to V6.x through modularization, UI layer remained stable while data files expanded from 0 to 1,250 lines and documentation from ~300 to 900 lines. Total increase represents intentional complexity relocation from hard-coded logic to maintainable data structures and comprehensive documentation supporting production deployment, see table 2 for comparison V1.x to V6.x code

lines and functionalities.

Table 2: Code lines and details

Version	Key Features	Code lines	Comments
V1	Proof of concepts	~250	Initial prototype, single file, functionality in one file
V2	Basic UI	~400	V1 extended, two-files managing 3DX connexion and PyQt5 user interface
V3	Pattern Matching	~800	First optimised iteration, integrator working with three modules, Python and JSON files; CAD function capabilities
V4	Geometry complete	~1200	CAD functionalities extended, Wireframe, Surfaces and Solid operations, UI optimised
V5	Hybrid System & JSON libraries	~1500	Performance optimisation, learning engine introduction, history navigation, help in context, API, material & rules library extension
V6	Modular system & JSON libraries	~1800 *~1500 (JSON)*	Modular architecture, code-date separation, CAD capabilities, command extension, test loop, material & rules flexible and self-learning capabilities

4. Discussion

The three-tier execution strategy successfully balances competing requirements of performance, flexibility, and reliability. Learned commands provide optimal performance by eliminating both pattern matching and network overhead, capitalizing on the observation that users repeatedly execute small command subsets. Static pattern matching serves as reliable fallback for predictable commands not yet learned, delivering substantial improvement over LLM calls while enabling sophisticated parameter extraction through regex without AI inference overhead. LLM integration as final tier preserves flexibility for novel inputs despite higher latency, with automatic learning converting initial processing into future cache hits and creating self-improving behaviour over time.

This hybrid approach diverges from pure LLM systems like AutoCAD AI [16] Assistant that query models for every command and pure pattern-matching systems like traditional command parsers (limited flexibility, no natural language understanding). The graduated strategy captures advantages of both paradigms while mitigating their respective weaknesses through intelligent tier selection based on command recognition confidence.

The modular data architecture addresses fundamental software engineering principles through separation of concerns. Domain knowledge (materials, manufacturing rules) evolves independently from algorithmic logic, yet traditional CAD systems intermingle these aspects.

Platform-specific COM-API integration with 3DX prioritizes depth over breadth, providing full access to parametric features and real-time creation capabilities rather than basic geometry export across multiple platforms. This design decision enabled rapid prototyping and real-world validation but imposes Windows-only operation, version-dependent API stability, and single-application targeting. Future generalization strategies include abstracting CAD operations behind interface layers, implementing adapter patterns for multiple backends, and considering intermediate tier local-APIs for cross-platform support.

The implemented operations represent foundational capabilities demonstrating architectural feasibility rather than comprehensive coverage. Notable omissions include advanced curves (splines, conics, projections), surface operations (sweeps, lofts, boundaries), feature operations (fillets, holes, patterns, shells), and assembly operations (components, constraints, mates). Expansion priority targets complex tasks such as "boundary box generation, automated colouring, export as STP, constant thickness check".

Comparison with related work reveals distinct contributions. AutoCAD AI employs pure LLM-based interpretation with integrated help but without learning capabilities for performance optimization. Arena AI [17] provides cloud-based LLM integration with cross-platform collaboration but requires persistent internet connectivity.

This work distinguishes itself through the hybrid three-tier architecture with self-learning, modular knowledge base enabling non-programmer extension, integrated materials library with design rules (typically external databases), and production-ready implementation with comprehensive error handling, testing, and documentation. The combination of natural language flexibility with performance optimization through learning represents a novel balance does not present in existing commercial or academic CAD interfaces.

Natural language interfaces demonstrate potential to reduce CAD learning curves by estimated 50-60% in time-to-productivity, particularly benefiting engineering students (faster project completion), occasional users (project managers, sales engineers without deep CAD expertise), and rapid prototyping scenarios (quick concept validation). This pattern of knowledge-integrated design tools represents a valuable direction for future CAD system development, where manufacturing knowledge, contextual guidance, user-extensible rules, and organizational knowledge capture become first-class system features rather than external references.

The preliminary results from this early-stage research demonstrate feasibility and establish baseline performance characteristics. However, extensive longitudinal studies with diverse user populations will be necessary to validate learning curve reduction estimates and establish statistical significance for productivity improvements. Current findings derive from limited testing scenarios and should be interpreted as proof-of-concept rather than definitive performance claims.

5. Conclusion

This research successfully developed and validated prompt-to-CAD, a self-learning natural language with a high command success rate. The modular design separates code logic from domain data, enabling code-free extension of materials and design rules by non-programmers.

The system addresses key research questions through preliminary validation. Natural language interfaces can reduce CAD learning barriers while maintaining professional functionality through success rate and flexible command expression, though extensive user studies remain necessary to quantify learning curve improvements. The graduated execution strategy with self-learning demonstrates efficiency gains through caching, with break-even occurring after 2-3 command repetitions in observed usage patterns. Domain knowledge integrates effectively via JSON-based modular architecture, validated through comprehensive materials (density, strength, applications, design tips) and rules (draft angles, tolerances, manufacturability guidelines). Optimal architectural patterns combine deterministic execution for common operations with AI flexibility for novel inputs and automatic learning for continuous improvement.

Practical applications span engineering education (accelerated training, reduced cognitive load), rapid prototyping (quick validation without deep expertise), production workflows (reduced routine operation time, embedded manufacturability checking), and research (design automation studies, human-AI collaboration). Future work directions include geometry expansion (fillets, holes, patterns, sweeps), knowledge base growth (additional materials, cost estimation, manufacturability scoring), multi-modal input (voice, sketches, images), cross-platform support (adapter layers, CAD-agnostic representations), advanced AI capabilities (conversational clarification, context maintenance, design suggestions), and longitudinal user studies quantifying productivity improvements.

This work contributes a reusable architectural pattern for AI-assisted technical tools, demonstrating that hybrid approaches combining deterministic reliability with AI flexibility can achieve both performance optimization and natural language understanding. The system establishes that AI can lower barriers to sophisticated technical tools without sacrificing professional capabilities, though this early-stage research requires substantial additional validation before definitive claims regarding productivity improvements can be made.

5.1 Limitations

implementation achieved production-ready status within a constrained technological environment, specifically limited to Windows operating systems due to the inherent architecture of Component Object Model (COM) API technology, which serves as the exclusive interface for programmatic control of 3DX platform. This platform dependency restricts deployment scenarios to Windows-

based workstations and precludes adoption on macOS or Linux systems commonly used in academic research environments and certain industrial contexts. The strategic decision to target 3DX exclusively enabled deep integration with a professional-grade CAD system widely deployed in automotive and aerospace industries, validating the approach on complex, industry-standard software rather than simplified or academic CAD tools. However, this single-platform focus presents significant generalization challenges for future research, as extending the system to alternative CAD platforms such as SolidWorks, Creo, Fusion 360, or open-source alternatives like FreeCAD would require substantial architectural modifications including development of platform-specific adapters, abstraction of geometric operations behind a unified interface layer, and potential reimplementation of the communication mechanism to accommodate different automation APIs (COM, REST, Python bindings). The fundamental technical challenge lies not merely in replicating the interface across multiple platforms, but in reconciling fundamentally different geometric modelling paradigms, varying feature parametrization schemes, and inconsistent API stability guarantees across commercial CAD vendors, maintaining proprietary automation interfaces with limited cross-platform standardization.

Acknowledgments:

For this research, the used tools are listed below: (1) Personal laptop capable of AI processing with CUDA enabled; (2) Python 3.12 locally installed; (3) local environment set-up (PyQt5, local AI, APIs); (4) 3DXPERIENCE 2025X – Student license with limited functionality (5) Claude API credit for API calls, (6) Python SDK simplified integration abstracting HTTP communication details and response parsing complexity, (7) pywin32 - Python for Windows Extensions enabling Windows COM interface access essential for 3DX automation, (8) Python standard library contributors for essential modules: JSON for data serialization, re for regular expression pattern matching, datetime for timestamp generation, difflib for fuzzy string matching implementing Sequence Matcher algorithm, and os/sys for file system operations and environment configuration, (9) Development Tools: Git version control system enabling iterative development with complete history tracking. Visual Studio Code and PyCharm IDEs providing Python development support with debugging, code completion, and refactoring capabilities, (10) Design and Manufacturing [13] , Plastic Parts Design for Injection Moulding [12], Manufacturer material data sheets from BASF [18], Sabic [19], DuPont [20] and others supplying materials properties data, (11) Open Source Community: Broader open source ecosystem providing tools, libraries, and knowledge resources supporting rapid development without requiring extensive infrastructure investment. Stack Overflow community for troubleshooting assistance. GitHub for example code and architectural patterns & CADAM [21] system as open-source data for CAD generation.

References

[1] Li, Kun-Ying, Cheng-Kai Huang, Qing-Wei Chen, Hsuan-Cheng Zhang, and Tsann-Tay Tang. "Generative AI and CAD Automation for Diverse and Novel Mechanical Component Designs under Data Constraints." *Discover Applied Sciences* 7 (2025): 315. https://doi.org/10.1007/s42452-025-06833-5.

[2] Xavier, Amatriain. "Prompt Design and Engineering: Introduction and Advanced Methods." arXiv preprint, January 2024. https://arxiv.org/pdf/2401.14423.

[3] Anthropic. "Introducing Claude Sonnet 4.5." Anthropic AI News, 2025. Accessed November, 2025. https://www.anthropic.com/news/claude-sonnet-4-5.

[4] OpenAI. "Introducing GPT-5." OpenAI Blog, 2025. Accessed November, 2025. https://openai.com/index/introducing-gpt-5/.

[5] Nielsen, Jakob. "Response Times: The 3 Important Limits." Nielsen Norman Group, January 1, 1993. Accessed November, 2025. https://www.nngroup.com/articles/response-times-3-important-limits/.

[6] Python Software Foundation. "Python 3.12.12 Documentation." 2025. Accessed November, 2025. https://docs.python.org/release/3.12.12/.

[7] JSON.org. "Introducing JSON." 2025. Accessed November, 2025. https://www.json.org/json-en.html.

[8] Qt Company. "Qt for Python Documentation." The Qt Company Ltd., 2023. Accessed November, 2025. https://doc.qt.io/qtforpython/.

[9] Anthropic. "Build with Claude: Using the Messages API." Claude Documentation, 2025. Accessed November, 2025. https://docs.claude.com/en/docs/build-with-claude/working-with-messages.

[10] Dassault Systèmes. "3DEXPERIENCE User Assistance 2025x." Dassault Systèmes SE, 2025. Accessed November, 2025. https://help.3ds.com/2025x/English/DSDoc/FrontmatterMap/DSDocHome.htm?contextscope=cloud&redirect_lang=English.

[11] Dassault Systèmes. "Developer Assistance R2025x API Reference." Dassault Systèmes SE, 2025. Accessed November, 2025. https://media.3ds.com/support/documentation/developer/Cloud/en/DSDoc.htm?show=CAADocQuickRefs/DSDocHome.htm.
[12] Malloy, Robert A. *Plastic Part Design for Injection Molding: An Introduction*. 2nd ed. Munich, Hanser Publications, 2010.
[13] Bralla, James G. (ed.). *Design for Manufacturability Handbook*. 2nd ed. New York, McGraw-Hill Professional, 1998.
[14] pywin32 Contributors. "Python for Windows Extensions (pywin32)." Python Package Index, 2025. Accessed January 15, 2025. https://pypi.org/project/pywin32/.
[15] MATER. "Catalog Online: Materials Database." MATER Library, 2025. Accessed November, 2025. https://app.materlibrary.ro/ro/materiale.
[16] Autodesk Inc. "AutoCAD 2024 AI Assistant Features." AutoCAD 2024 Product Documentation, 2024. Accessed November, 2025. https://help.autodesk.com/view/ACD/2024/ENU/?guid=GUID-B4E1E636-E08E-4277-8971-910D47440116.
[17] PTC. "PTC Launches Arena AI Assistant to Accelerate PLM and QMS Workflows." PTC Press Release, 2025. Accessed January 15, 2025. https://www.ptc.com/en/news/2025/ptc-launches-arena-ai-assistant.
[18] BASF SE. "Engineering Plastics: Performance Polymers." BASF Plastics & Rubber, 2025. Accessed November, 2025. https://plastics-rubber.basf.com/global/en/performance_polymers#item-1705923019047-1853604672.
[19] SABIC. "Polymers." SABIC Global, 2025. Accessed November, 2025. https://www.sabic.com/en/products/polymers.
[20] DuPont, Inc. "Interior Solutions: Automotive Applications." DuPont Mobility & Materials, 2025. Accessed November, 2025. https://www.dupont.com/mobility/automotive/interior.html.
[21] CADAM. "ADAM Creation: AI-Powered Design Platform." 2025. Accessed November, 2025. https://adam.new/app/editor/7d4c0720-6ccc-46df-a0f3-0993c1e1eac5.

Abbreviations
CAD – Computer-Aided Design
AI – Artificial Intelligence
ML – Machine Learning
LLM – Large Language Model
3DX – 3DEXPERIENCE R2025x (CATIA v6) – Dassault Systems
NLP – Natural Language Processing
JSON – JavaScript Object Notation
GUI – Graphical User Interface
API – Application Programming Interface
COM – Component Object Model
SDK – Software Development Kit
DC01 – Mild Steel
PP – Polypropylene
PA – Polyamide (Nylon)
GF – Glass Filled
SPI – Society of the Plastics Industry
V1x to V6.x – Version 1 through Version 6
STP – Standard for the Exchange of Product Model Data
VBA – Visual Basic for Applications

Appendix A - System's Architecture

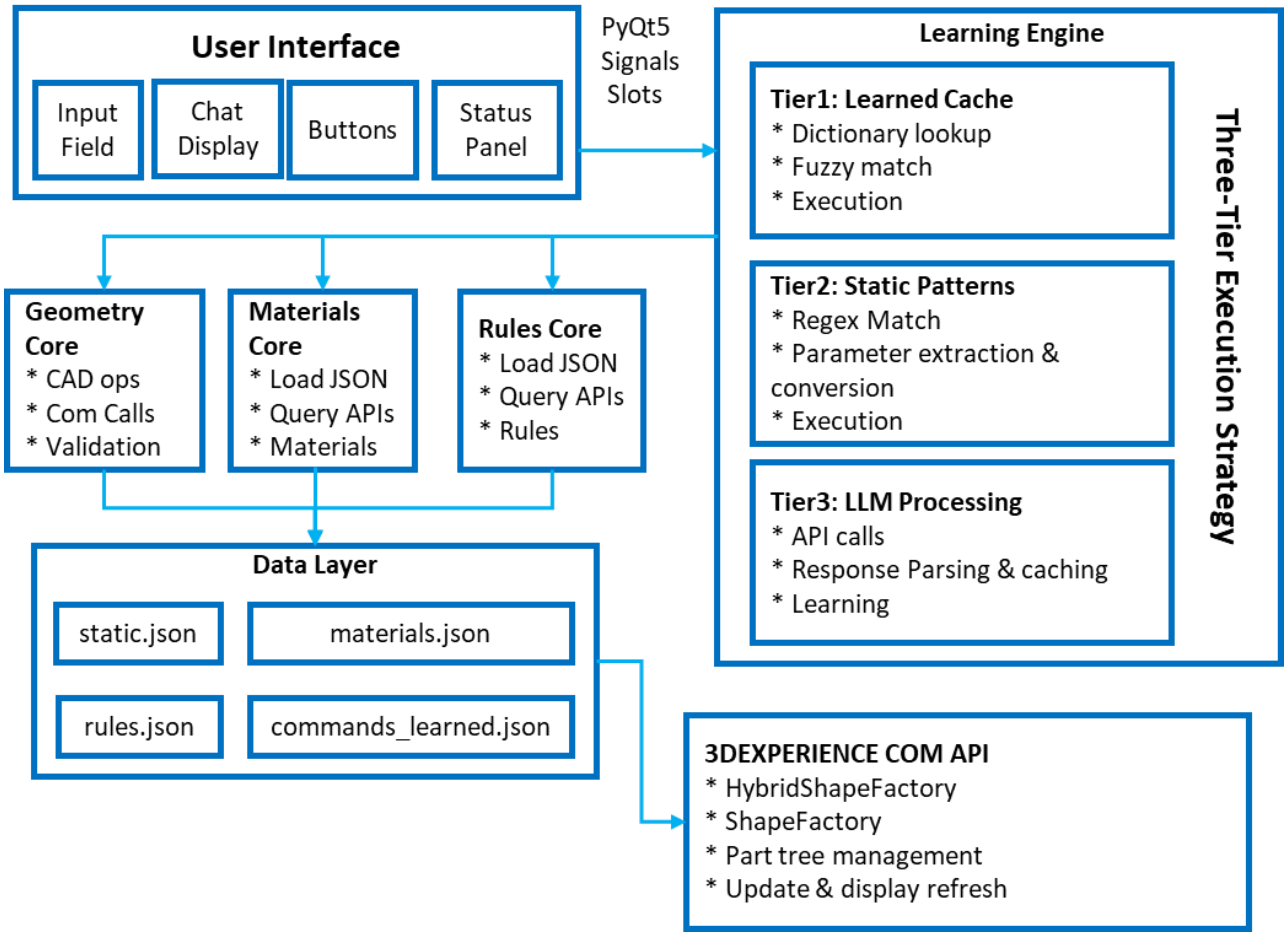

Fig. 4. System's Architecture & Logic

Appendix B – Terminal display data

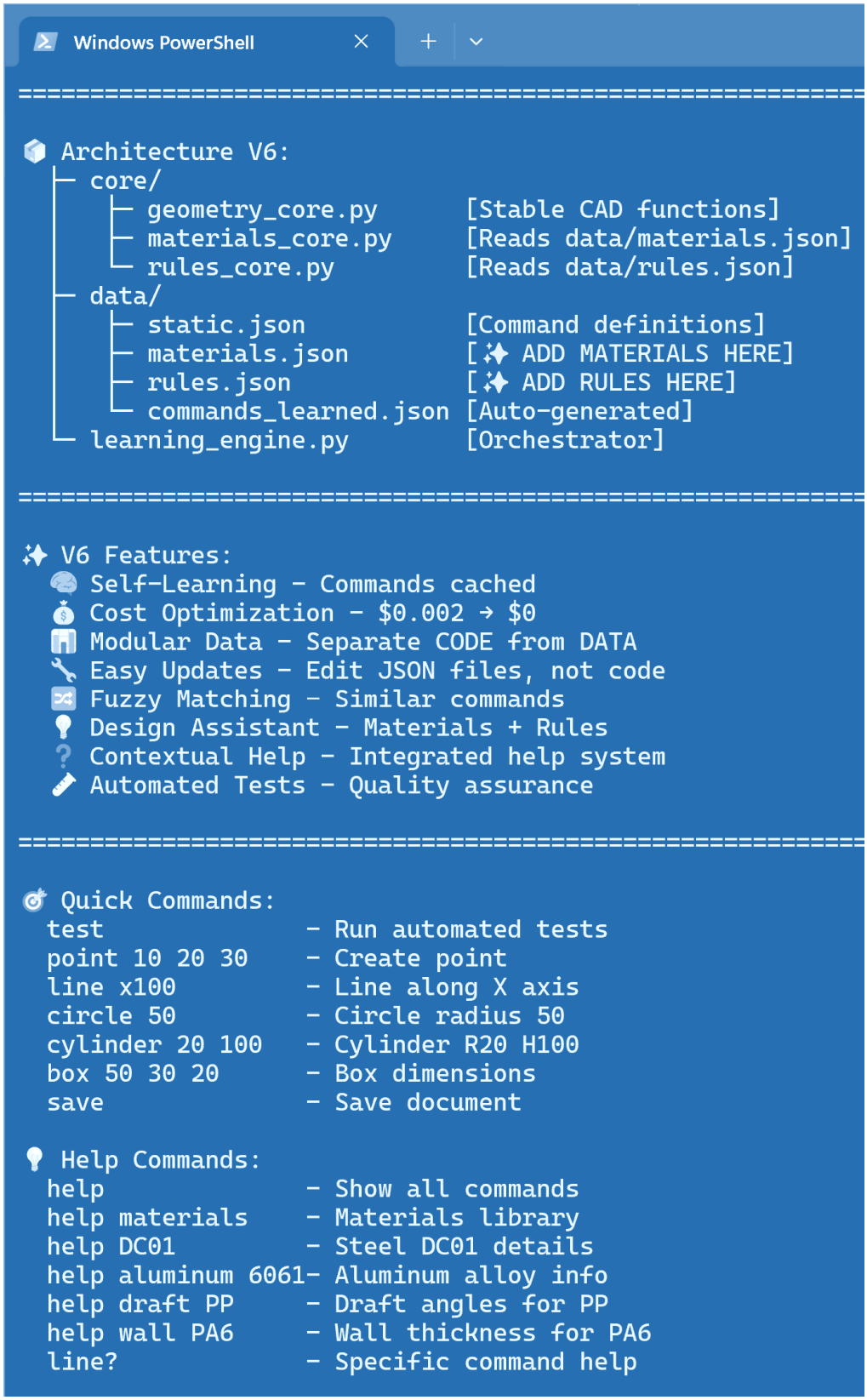

Fig. 5. Terminal information implemented for V5.x-V6.x

Appendix C – Program examples (type: main_V6.py)

```
main_V6.py
Text-to-CAD V6 - Modular Architecture
"""
import sys
from PyQt5.QtWidgets import QApplication, QInputDialog
from ui_interface_V6 import TextToCADInterface


def main():
    print("=" * 70)
    print("🚀 Text-to-CAD V6.0 - MODULAR ARCHITECTURE")
    print("=" * 70)
    print()
    print("📦 Architecture V6:")
```

Fig. 6. Code as example: main_V6.py

```
ui_interface_V6.py
"""

from PyQt5.QtWidgets import *
from PyQt5.QtCore import Qt, QDateTime, pyqtSignal
from PyQt5.QtGui import QFont, QTextCursor, QKeyEvent
import win32com.client
from learning_engine import LearningEngine


class EnterTextEdit(QTextEdit):
    enter_pressed = pyqtSignal()

    def __init__(self, parent=None):
        super().__init__(parent)
        self.history = []
        self.history_idx = -1
        self.current_text = ""

    def keyPressEvent(self, event: QKeyEvent):
```

Fig. 7. Code as example: ui_interface_V6.py

Learning engine

```
"""
learning_engine.py - V6.2 - Materials Intelligence
Self-learning + Help + Design Assistant + Materials AI
"""

import json
import re
from datetime import datetime
from difflib import SequenceMatcher
import os
import sys

sys.path.insert(0, os.path.join(os.path.dirname(__file__), 'core'))

from geometry_core import GeometryCore
from materials_core import MaterialsCore
from rules_core import RulesCore


class LearningEngine:

    CLAUDE_MODEL = "claude-sonnet-4-20250514"

    def __init__(self, claude_api_key=None):
        self.version = "6.2"
        self.claude_api_key = claude_api_key
```

Fig. 8. Code as example: learning_engine.py

Appendix D – Commands list

General Commands
1. `save` - Save document via part update
2. `test` - Run automated test suite (15 operations)

Wireframe Commands:
3. `point X Y Z` - Create point at coordinates
   - Example: `point 10 20 30`
4. `edit point X Y Z to X2 Y2 Z2` - Modify point coordinates
   - Example: `edit point 10 20 30 to 15 25 35`
5. `line xLENGTH` - Line along X axis from origin
   - Example: `line x100`
6. `circle at point X Y Z PLANE RADIUS` - Circle centered at point
   - Example: `circle at point 10 20 30 xy 25`

Surface Commands:
7. `plane PLANE offset DISTANCE` - Offset plane from origin
   - Example: `plane xy offset 100`
8. `plane PLANE through point X Y Z` - Plane through existing point
   - Example: `plane xy through point 10 20 30`

Sketch Commands:
9. `rectangle WIDTH HEIGHT` - Rectangle sketch on XY plane
   - Example: `rectangle 50 30`

Solid Commands:
10. `cylinder RADIUS HEIGHT` - Cylindrical solid via pad
   - Example: `cylinder 20 100`
11. `box LENGTH WIDTH HEIGHT` - Rectangular solid via pad
   - Example: `box 50 30 20`

Help Commands
12. `help` - General command overview
13. `help materials` - List all materials
14. `help MATERIAL` - Specific material info
   - Example: `help DC01`, *
15. `help draft [MATERIAL]` - Draft angle recommendations
   - Example: `help draft`, `help draft PP`
16. `help wall [MATERIAL]` - Wall thickness guidelines
   - Example: `help wall`, `help wall PA6`
17. `COMMAND?` - Command-specific help
   - Example: `line?`, `cylinder?`

Natural Language:
- Any descriptive request interpreted by Claude API
- Examples: "create a point at 10 20 30", "make me a box 50 by 30 by 20"
- Automatically learned after first successful execution

Material Database - Sample

Complete Material Entry Example (DC01 Steel):

```json
"DC01": {
  "name": "DC01 Cold Rolled Steel",
  "category": "Metals",
  "type": "Cold rolled, deep drawing quality",
  "aliases": ["DC 01", "1.0330"],
  "properties": {
    "density": "7.85 g/cm³",
    "yield_strength": "140-300 MPa",
    "tensile_strength": "270-410 MPa",
    "elongation": "≥28%"
  },
  "characteristics": [
    "Excellent formability",
    "Good weldability",
    "Surface quality for painting/coating",
    "Low carbon content"
  ],
  "applications": [
    "Sheet metal parts",
    "Automotive body panels",
    "Appliance housings",
    "Deep drawn components",
    "Brackets and mounting plates"
  ],
  "design_tips": [
    "Min bend radius: 0.5-1× thickness",
    "Ideal for complex forming operations",
    "Requires surface protection (paint/coating)",
    "Good for welding and spot welding",
    "Standard thickness range: 0.5-3.0mm"
  ]
}
```

Fig. 9. Example of Material Data Sheet (sample data)

Appendix E – Design Rule summary

```
**Draft Angles Quick Reference:**
| Material | Exterior | Interior | Notes |
|----------|----------|----------|-------|
| PP | 1-4° | 2-5° | Good release |
| PP-GF30 | 1.5-6° | 2-6° | More draft needed |
| PA6 | 1-5° | 2-5° | Varies with moisture |
```

```
**Bend Radius Quick Reference (Sheet Metal):**
| Material | Minimum | Standard | Notes |
|----------|---------|----------|-------|
| DC01 Steel | 0.5× t | 1× t | Excellent formability |
| S235JR Steel | 1× t | 1.5× t | Moderate formability |
```

Fig. 10. Example of Material Data Sheet (sample data)