Kognitives Scaffolding: Eine Solver-gestützte, Multi-Agenten-Architektur für kontextsensitive KI in FreeCAD

Abschnitt 1: Das FreeCAD-Ökosystem: Ein parametrisches, erweiterbares Ziel für die KI-Integration

Die erfolgreiche Integration eines KI-gestützten Addons in eine bestehende Softwareumgebung erfordert ein tiefgreifendes Verständnis der grundlegenden Architektur, der Designphilosophie und der programmatischen Schnittstellen dieser Umgebung. FreeCAD, als Open-Source-Computer-Aided-Design (CAD)-Software, stellt ein besonders faszinierendes und herausforderndes Ziel dar. Seine Architektur ist nicht nur ein Werkzeugkasten für geometrische Operationen, sondern verkörpert eine spezifische Methodik des Designs: das parametrische Modellieren. Diese Methodik, die auf dem leistungsstarken Open CASCADE Technology (OCCT) Geometriekern aufbaut, diktiert die Art und Weise, wie ein KI-Agent "denken" und handeln muss, um sinnvolle und gültige 3D-Modelle zu erstellen. Ein Agent, der in diesem Ökosystem operieren soll, muss lernen, in Begriffen von Merkmalen, Abhängigkeiten und lösbaren Zwangsbedingungen zu agieren, anstatt sich auf die bloße Manipulation von geometrischen Koordinaten zu beschränken. Dieser Abschnitt analysiert die fundamentalen Säulen des FreeCAD-Ökosystems – das parametrische Paradigma, die Python-Scripting-API und den integrierten Constraint-Solver – um die grundlegenden Anforderungen und Möglichkeiten für einen intelligenten KI-Assistenten zu definieren.

1.1. Das parametrische Designparadigma und der Open CASCADE Kernel

Das Herzstück von FreeCAD ist sein Bekenntnis zum parametrischen Modellieren. Im

Gegensatz zur direkten Modellierung, bei der Geometrien direkt manipuliert werden (z. B. das Ziehen einer Fläche), sind in FreeCAD alle Objekte nativ parametrisch.[1] Das bedeutet, ihre Form wird nicht durch eine statische Anordnung von Punkten und Flächen definiert, sondern durch eine Abfolge von Operationen und deren zugehörigen Parametern. Ein einfacher Würfel ist nicht nur eine Sammlung von sechs Flächen und acht Eckpunkten; er ist das Ergebnis einer

Part::Box-Funktion, die durch die Eigenschaften Länge, Breite und Höhe definiert wird. Jede Änderung dieser Eigenschaften führt zu einer Neuberechnung des Modells, wobei eine präzise Modellierungshistorie erhalten bleibt.[1] Diese Historie ist nicht nur eine Aufzeichnung, sondern das eigentliche Wesen des Modells.

Die technische Grundlage für diese Funktionalität liefert der Open CASCADE Technology (OCCT) Geometriekern, eine hochentwickelte C++-Bibliothek, die auf die Modellierung von Volumenkörpern spezialisiert ist.[1] Die primäre Datenstruktur, die im Part-Modul von FreeCAD verwendet wird, ist der B-Rep-Datentyp (Boundary Representation) von OCCT.[3] Eine B-Rep-Darstellung definiert einen Volumenkörper durch seine topologischen Elemente: Ecken (Vertices), Kanten (Edges), Flächen (Faces), Hüllen (Shells) und den Volumenkörper (Solid) selbst. Diese Elemente sind hierarchisch und relational miteinander verbunden. Eine Fläche wird durch eine Schleife von Kanten begrenzt, und eine Kante wird durch zwei Ecken definiert. Diese topologische Information ist entscheidend und unterscheidet B-Rep-Modelle fundamental von anderen 3D-Darstellungen wie Polygonnetzen (Meshes) oder Punktwolken, die in vielen anderen KI-Anwendungen für 3D-Geometrie üblich sind.

Für einen KI-Agenten hat dies tiefgreifende Implikationen. Ein KI-Modell, das darauf trainiert ist, Polygonnetze zu erzeugen, wie es in der generativen KI für Spiele oder visuelle Effekte üblich ist, ist für die CAD-Welt ungeeignet. Ein solches Modell würde eine "dumme" Geometrie ohne die für die Fertigung und Bearbeitung notwendige Präzision und topologische Integrität erzeugen. Ein KI-Agent für FreeCAD kann nicht einfach eine fertige Form ausgeben; er muss eine prozedurale Sequenz von gültigen FreeCAD-Operationen generieren, die bei ihrer Ausführung ein valides, vielfältiges (manifold) B-Rep-Volumenkörpermodell ergeben. Der Fokus verschiebt sich von einem deklarativen ("was") zu einem prozeduralen ("wie") Ansatz. Die KI muss den Konstruktionsprozess selbst erlernen und nachbilden.

1.2. Die Python Scripting API: Die Schnittstelle der KI zur Welt

Während der Kern von FreeCAD aus Gründen der Robustheit und Leistung in C++ geschrieben ist, sind große Teile der äußeren Schichten, der Workbenches und fast die gesamte Kommunikation zwischen dem Kern und der Benutzeroberfläche in Python implementiert.[1] Diese architektonische Entscheidung macht die Python-Scripting-API zum alleinigen und mächtigen Mechanismus, durch den ein KI-Agent die FreeCAD-Umgebung wahrnehmen und in ihr handeln kann. Jede Aktion, die ein menschlicher Benutzer über die grafische Oberfläche ausführt – vom Erstellen eines Würfels bis zum Anwenden einer komplexen Zwangsbedingung – entspricht einem oder mehreren Python-Befehlen, die im Hintergrund ausgeführt werden.[4]

Für die Entwicklung eines KI-Addons sind insbesondere zwei Workbenches und ihre APIs von zentraler Bedeutung:

Part Workbench: Diese Workbench bietet die grundlegenden Werkzeuge zur Erstellung und Manipulation von Volumenkörpern (Solid Shapes). Die API ermöglicht die skriptgesteuerte Erzeugung von geometrischen Primitiven und die Durchführung von booleschen Operationen, die die Grundlage der konstruktiven Festkörpergeometrie (Constructive Solid Geometry, CSG) bilden.[3]

- **Primitive erstellen:** Ein einfacher Würfel und ein Zylinder können wie folgt erstellt werden:
  ```
  Python
  import FreeCAD as App
  import Part
  doc = App.activeDocument()

  # Erstellt einen Würfel
  box = doc.addObject("Part::Box", "MyBox")
  box.Length = 10.0
  box.Width = 10.0
  box.Height = 10.0

  # Erstellt einen Zylinder
  cylinder = doc.addObject("Part::Cylinder", "MyCylinder")
  cylinder.Radius = 5.0
  cylinder.Height = 20.0
  cylinder.Placement.Base = App.Vector(5, 5, 0) # Positioniert den Zylinder
  ```

- **Boolesche Operationen:** Diese Operationen kombinieren zwei Formen zu einer

neuen. Die drei grundlegenden Operationen sind Vereinigung (fuse), Differenz (cut) und Schnittmenge (common).[1]

```python
# Differenz: Zieht den Zylinder vom Würfel ab
cut_obj = doc.addObject("Part::Cut", "Cut")
cut_obj.Base = box
cut_obj.Tool = cylinder
```

Sketcher Workbench: Dies ist das Herzstück des parametrischen Designs in FreeCAD.[1] Ein Sketch ist eine 2D-Zeichnung, die aus geometrischen Elementen (Linien, Kreise, Bögen) und Zwangsbedingungen (Constraints) besteht, die die Beziehungen zwischen diesen Elementen definieren. Ein "vollständig bestimmter" (fully constrained) Sketch hat keine Freiheitsgrade mehr; seine Form ist eindeutig durch die Zwangsbedingungen definiert.[7] Diese Skizzen dienen als Basis für 3D-Operationen wie Extrusion (Pad) oder Rotation (Revolve).

- **Sketch erstellen und Geometrie hinzufügen:**

```python
import Sketcher

# Erstellt ein neues Sketch-Objekt auf der XY-Ebene
my_sketch = doc.addObject('Sketcher::SketchObject', 'MySketch')
my_sketch.Placement.Base = App.Vector(0, 0, 0)

# Fügt eine Linie und einen Bogen hinzu
line_idx = my_sketch.addGeometry(Part.LineSegment(App.Vector(0,0,0),
App.Vector(10,0,0)), False)
arc_idx =
my_sketch.addGeometry(Part.ArcOfCircle(Part.Circle(App.Vector(10,10,0),
App.Vector(0,0,1), 10), 3.14159, 0), False)
```

- **Zwangsbedingungen anwenden:** Dies ist der entscheidende Schritt, um Intelligenz in die Skizze zu bringen. Jede Zwangsbedingung wird durch ihren Typ und die Indizes der betroffenen geometrischen Elemente und deren Punkte definiert.[8]

```python
# Macht die Linie horizontal
my_sketch.addConstraint(Sketcher.Constraint('Horizontal', line_idx))

# Macht den Endpunkt der Linie und den Startpunkt des Bogens deckungsgleich (Coincident)
```

```
# Parameter: Typ, Index_Geometrie1, Punkt_Index1, Index_Geometrie2, Punkt_Index2
my_sketch.addConstraint(Sketcher.Constraint('Coincident', line_idx, 2, arc_idx, 1))

# Definiert die Länge der Linie auf 50 Einheiten
my_sketch.addConstraint(Sketcher.Constraint('Distance', line_idx, 50))
```

Die Struktur des Dokuments selbst wird ebenfalls über die API verwaltet. App.newDocument() oder App.activeDocument() bieten den Zugriff auf das aktuelle Arbeitsdokument.[3] Jedes hinzugefügte Objekt (

doc.addObject(...)) hat Eigenschaften wie Shape (die geometrische Darstellung) und Placement (Position und Orientierung im Raum), die programmatisch gelesen und verändert werden können.[4] Der Befehl

doc.recompute() ist von entscheidender Bedeutung: Er weist FreeCAD an, die gesamte Objekthistorie neu zu bewerten und das Modell basierend auf den neuesten Parameter- und Zwangsbedingungsänderungen zu aktualisieren. Er ist der Trigger, der den Solver aktiviert und die parametrischen Änderungen sichtbar macht.[3]

1.3. Der Constraint-Solver: Das wahre geometrische "Gehirn"

Ein häufiges Missverständnis bei der Konzeption eines KI-Systems für CAD ist die Annahme, die KI müsse komplexe geometrische Berechnungen selbst durchführen. Dies ist jedoch nicht der Fall und wäre auch ein ineffizienter Ansatz. FreeCAD verfügt über einen hochentwickelten, integrierten Constraint-Solver, der das mathematische Herzstück des Sketchers bildet.[1] Wenn ein Benutzer oder ein Skript eine Reihe von geometrischen Elementen und Zwangsbedingungen definiert, ist es die Aufgabe des Solvers, dieses System von Gleichungen und Ungleichungen zu lösen, um die exakten Koordinaten aller Punkte zu bestimmen.

Die Rolle der KI ist daher nicht die eines geometrischen Problemlösers, sondern die eines präzisen Problemformulierers. Die Intelligenz des KI-Agenten liegt darin, die oft vage und semantisch formulierte Absicht des Benutzers (z. B. "Zeichne ein Rechteck, das an diesem Kreis anliegt") in eine eindeutige, formal korrekte Spezifikation für den Solver zu übersetzen. Dies bedeutet, die richtigen geometrischen Primitive zu erstellen (Part.LineSegment) und die korrekten Zwangsbedingungen (Constraint('Tangent',...)) anzuwenden. Diese kognitive Lastenverteilung ist der

Schlüssel zu einer eleganten und robusten Lösung. Sie nutzt die Stärken beider Systeme: die Fähigkeit des großen Sprachmodells (LLM), natürliche Sprache und Absichten zu verstehen, und die Fähigkeit des spezialisierten, deterministischen CAD-Solvers, geometrische Probleme mit absoluter Präzision zu lösen. Der Versuch, dem LLM geometrisches Schließen beizubringen, ein Bereich, in dem es bekanntermaßen schwach ist [11], während ein perfektes Werkzeug für genau diese Aufgabe bereits zur Verfügung steht, wäre ein grundlegender architektonischer Fehler.

Die parametrische Natur von FreeCAD führt auch zu einer kritischen Eigenschaft des Systems: Die Modellhistorie ist ein sequentielles, zustandsabhängiges Programm. Jede Operation baut auf dem Zustand auf, den die vorhergehenden Operationen geschaffen haben. Dies manifestiert sich im berüchtigten "Topological Naming Problem".[12] Wenn beispielsweise eine Kante für eine Verrundungsoperation (Fillet) ausgewählt wird, referenziert FreeCAD sie intern mit einem Namen wie "Edge8". Wenn eine frühere Operation in der Historie (z. B. die Basisskizze) so geändert wird, dass sich die Topologie des Körpers ändert, kann "Edge8" entweder nicht mehr existieren oder sich auf eine völlig andere Kante beziehen, was dazu führt, dass die Verrundungsoperation fehlschlägt.

Dies zeigt, dass ein KI-Agent Befehle nicht isoliert generieren kann. Er muss ein präzises internes Modell des aktuellen Dokumentenzustands – einschließlich der vorhandenen Objekte, ihrer Topologie und der Skizzen – aufrechterhalten, um den *nächsten gültigen* Befehl zu generieren. Die Verwaltung dieses Kontexts ist keine optionale Zusatzfunktion, sondern eine unabdingbare architektonische Notwendigkeit für jede Form von sinnvoller, schrittweiser Modellgenerierung.

Abschnitt 2: Die semantische Lücke: Kernherausforderungen für generative KI im parametrischen CAD

Die Anwendung generativer KI-Modelle, insbesondere großer Sprachmodelle (LLMs), auf das prozedurale 3D-Modellieren in einer parametrischen CAD-Umgebung wie FreeCAD stößt auf eine Reihe fundamentaler Herausforderungen. Diese lassen sich unter dem Begriff der "semantischen Lücke" zusammenfassen: die Kluft zwischen der hochrangigen, oft mehrdeutigen und absichtsbasierten Sprache des menschlichen Designs und der niederrangigen, präzisen, geordneten und syntaktisch starren

Sprache der CAD-API. Die Überbrückung dieser Lücke erfordert mehr als eine einfache Text-zu-Code-Übersetzung; sie erfordert ein tiefes Verständnis von geometrischem Schließen, Präzision, langfristiger Planung und Designabsicht – alles Bereiche, in denen heutige LLMs erhebliche Schwächen aufweisen.

2.1. Das Scheitern des geometrischen Schließens

Die grundlegendste Dissonanz zwischen LLMs und CAD-Systemen liegt in ihrer Natur. LLMs sind probabilistische Modelle, die darauf trainiert sind, das wahrscheinlichste nächste Token in einer Sequenz vorherzusagen. Ihre Stärke liegt in der Erzeugung plausibler, menschenähnlicher Texte. CAD-Systeme hingegen sind deterministische Umgebungen, die auf mathematischer Exaktheit und absoluter Präzision beruhen, oft bis auf Mikrometer-Ebene.[1] Dieser fundamentale Gegensatz führt zu unweigerlichen Fehlern, wenn ein LLM versucht, direkt geometrische Aufgaben zu lösen.

Aktuelle Forschungsergebnisse bestätigen, dass LLMs Schwierigkeiten haben, prozedurale Geometrie zu erzeugen, was auf eine "Unfähigkeit zum räumlichen Schließen" zurückzuführen ist.[11] Sie besitzen kein angeborenes, verkörpertes Verständnis des dreidimensionalen Raums. Konzepte wie Tangentialität, Orthogonalität, Ausrichtung oder Kollisionsfreiheit sind für sie keine intuitiven Prinzipien, sondern Muster in Textdaten. Ein LLM kann zwar aus seinen Trainingsdaten lernen, dass die Zeichenfolge

Part.LineSegment(Vector(0,0,0), Vector(10,0,0)) eine horizontale Linie beschreibt, aber es "versteht" nicht die geometrische Bedeutung von Horizontalität. Dies führt dazu, dass es bei der Generierung von exakten Koordinaten für komplexe räumliche Anordnungen versagt. Die Wahrscheinlichkeit, dass ein LLM die exakten Gleitkommazahlen für einen Punkt generiert, der genau auf der Oberfläche eines Zylinders liegt, ist verschwindend gering. Dieses Problem wird durch die Tatsache verschärft, dass LLMs oft ein mangelndes kontextuelles Verständnis aufweisen und die spezifischen Zwecke, für die 3D-Modelle entworfen werden, nicht erfassen können.[15]

2.2. Die Herausforderung von Präzision und Zwangsbedingungen

Das Herzstück des parametrischen Modellierens ist nicht die Geometrie selbst, sondern das Netz von Zwangsbedingungen (Constraints), das ihre Beziehungen und ihr Verhalten definiert.[13] Eine KI muss nicht nur Linien und Kreise generieren, sondern auch die unsichtbaren Regeln, die sie steuern: diese Linie ist parallel zu jener, der Mittelpunkt dieses Kreises liegt auf jener Linie, der Abstand zwischen diesen beiden Punkten beträgt exakt 15.4 mm.

Die Erzeugung eines gültigen Satzes von Zwangsbedingungen ist eine komplexe Aufgabe. Selbst für menschliche Designer ist es eine häufige Fehlerquelle, widersprüchliche (over-constrained) oder redundante Zwangsbedingungen zu erstellen, die den Solver daran hindern, eine Lösung zu finden.[13] Eine KI muss lernen, einen minimalen und konsistenten Satz von Zwangsbedingungen zu generieren, der die Designabsicht vollständig erfasst, ohne den Sketch unlösbar zu machen. Dies erfordert ein Verständnis der Logik des Solvers und der Freiheitsgrade der Geometrie, was weit über die reine Syntax der

addConstraint-Funktion hinausgeht.

2.3. Langfristige Abhängigkeiten und Designabsicht

Ein CAD-Modell ist selten das Ergebnis einer kurzen, linearen Befehlsfolge. Es ist das Produkt eines langen Prozesses mit komplexen, nicht-lokalen Abhängigkeiten. Die Position eines Montagelochs, das in Schritt 50 erstellt wird, kann von einer grundlegenden Abmessung abhängen, die in der allerersten Skizze in Schritt 1 definiert wurde. LLMs, trotz ihrer immer länger werdenden Kontextfenster, haben bekanntermaßen Schwierigkeiten mit solch langfristiger Planung und der Aufrechterhaltung einer kohärenten "Absicht" über eine lange Sequenz von Generierungsschritten.[11] Dieses Problem, oft als "Lost in the Middle" bezeichnet, führt dazu, dass das Modell den übergeordneten Plan vergisst und sich auf lokale, unmittelbare Aufgaben konzentriert, was zu inkonsistenten oder fehlerhaften Modellen führt.[17]

Darüber hinaus ist die menschliche Designabsicht oft semantisch reich, aber prozedural mehrdeutig. Ein Befehl wie "Mach eine Halterung, um diesen Motor zu befestigen" ist für einen Menschen verständlich, für eine Maschine jedoch eine

enorme Herausforderung.[18] Die KI muss diese vage Anweisung in eine konkrete, geordnete Abfolge von CAD-Operationen übersetzen: Erstelle eine Grundplatte, füge einen vertikalen Flansch hinzu, schneide Befestigungslöcher mit dem richtigen Abstand basierend auf den Motorabmessungen, füge Versteifungsrippen für die Stabilität hinzu und verrunde scharfe Kanten, um die Spannungskonzentration zu reduzieren. Jeder dieser Schritte erfordert Domänenwissen und ein Verständnis der impliziten Anforderungen, die in der ursprünglichen Anweisung enthalten sind.[19] Das Problem ist also nicht nur die Übersetzung von Text in Code, sondern die Übersetzung von Absicht in einen strukturierten, prozeduralen Plan.

2.4. Datenknappheit und proprietäre Formate

Ein weiteres erhebliches Hindernis für die Entwicklung robuster KI-CAD-Systeme ist der Mangel an geeigneten Trainingsdaten. LLMs lernen aus riesigen Mengen von Text- und Codedaten. Im CAD-Bereich sind solche umfangreichen, qualitativ hochwertigen und öffentlich zugänglichen Datensätze jedoch eine Seltenheit.[21] Die meisten professionellen CAD-Modelle sind proprietär und in Unternehmen unter Verschluss. Öffentlich verfügbare Modelle sind oft nicht in einem prozeduralen Format (d. h. als Konstruktionshistorie oder Skript) gespeichert, sondern als statische Geometrie (z. B. STL-Dateien), die für das Erlernen des parametrischen Designprozesses ungeeignet ist.[23]

Diese Datenknappheit macht es schwierig, Modelle durch reines Supervised Learning auf realen Daten zu trainieren. Darüber hinaus erschweren Unterschiede in Dateiformaten, Modellierungsansätzen und Skriptsprachen zwischen verschiedenen CAD-Plattformen die Interoperabilität und die Erstellung standardisierter Datensätze, was eine nahtlose Integration in bestehende Ökosysteme zu einer großen Herausforderung macht.[21]

Die Konsequenz dieser Herausforderungen ist klar: Ein naiver Ansatz, der ein allgemeines LLM direkt mit der Aufgabe der FreeCAD-Skripterstellung betraut, ist zum Scheitern verurteilt. Das "Halluzinieren" von LLMs, das bei der Texterzeugung zu sachlichen Fehlern führt, manifestiert sich in der CAD-Welt als "geometrische Ungültigkeit". Ein LLM kann ein syntaktisch vollkommen korrektes FreeCAD-Python-Skript generieren, das bei der Ausführung dennoch ein geometrisch unsinniges oder ungültiges Ergebnis liefert – einen nicht-mannigfaltigen Volumenkörper, eine sich selbst schneidende Fläche oder eine unlösbare Skizze. Im

Gegensatz zu einem sachlichen Fehler in einem Text, den ein Mensch leicht erkennen kann, kann ein subtiler geometrischer Fehler visuell unbemerkt bleiben, aber zu katastrophalen Fehlschlägen in nachgelagerten Prozessen wie der Finite-Elemente-Analyse (FEA), dem 3D-Druck oder der CNC-Bearbeitung führen.[1] Dies impliziert, dass ein einfacher "Generieren und Ausführen"-Workflow unzureichend ist. Eine robuste Architektur muss zwingend eine Verifikations- und Feedbackschleife beinhalten, um diese geometrischen Halluzinationen abzufangen und zu korrigieren.

Abschnitt 3: Architekturen für kontextuelles Verständnis: Von RAG zu proaktivem Context Engineering

Die Überwindung der im vorherigen Abschnitt beschriebenen semantischen Lücke hängt entscheidend von der Fähigkeit des KI-Systems ab, den richtigen "Kontext" zur richtigen Zeit zu nutzen. Der Kontext ist das Arbeitsgedächtnis des Agenten, sein gesamtes Wahrnehmungsfeld, das ihm die Informationen liefert, die er benötigt, um eine fundierte Entscheidung für den nächsten Schritt zu treffen. Dieser Abschnitt untersucht die Entwicklung von Kontext-Management-Strategien, beginnend mit dem etablierten Ansatz der Retrieval-Augmented Generation (RAG) und fortschreitend zu einem anspruchsvolleren, proaktiven Context Engineering, das darauf abzielt, ein dynamisches "Bewusstsein" für den Agenten zu schaffen.

3.1. Baseline: Retrieval-Augmented Generation (RAG) für API-Wissen

Große Sprachmodelle (LLMs) leiden unter einer inhärenten Einschränkung: Ihr Wissen ist auf die Daten beschränkt, mit denen sie trainiert wurden, und wird mit der Zeit veraltet.[24] Retrieval-Augmented Generation (RAG) ist ein leistungsfähiger Rahmen, um dieses Problem zu lösen. Anstatt sich ausschließlich auf sein internes, statisches Wissen zu verlassen, wird dem LLM zur Inferenzzeit Zugriff auf eine externe, aktuelle Wissensdatenbank gewährt.[24]

In einer grundlegenden Implementierung für ein FreeCAD-AI-Addon würde ein RAG-System eine Vektordatenbank aufbauen, die aus der gesamten

FreeCAD-Python-API-Dokumentation [3], offiziellen und von der Community erstellten Tutorials [4] und einer Sammlung von Code-Beispielen besteht. Der Prozess funktioniert wie folgt:

1. **Indizierung:** Alle Dokumente werden in kleinere, semantisch kohärente Abschnitte (Chunks) zerlegt. Jeder Chunk wird dann mithilfe eines Embedding-Modells in einen hochdimensionalen Vektor umgewandelt. Diese Vektoren werden in einer Vektordatenbank gespeichert, die eine schnelle semantische Ähnlichkeitssuche ermöglicht.[24]
2. **Abruf:** Wenn der KI-Agent eine Aufgabe erhält (z. B. "Erstelle eine Verrundung an der Kante dieses Würfels"), wird die Benutzeranfrage ebenfalls in einen Vektor umgewandelt. Dieser Vektor wird verwendet, um die Vektordatenbank nach den relevantesten Chunks abzufragen – in diesem Fall wahrscheinlich die API-Dokumentation für die Part::Fillet-Funktion.
3. **Augmentation:** Die abgerufenen Dokumentations-Chunks werden zusammen mit der ursprünglichen Benutzeranfrage in den Prompt für das LLM eingefügt. Das LLM hat nun den genauen Kontext der relevanten API-Funktion, ihrer Parameter und Anwendungsbeispiele zur Verfügung, um einen korrekten Code zu generieren.

Dieser RAG-Ansatz ist eine notwendige Grundlage. Er reduziert grundlegende Halluzinationen, indem er das LLM mit Fakten "erdet" und sicherstellt, dass es die richtige Syntax und die richtigen Funktionsnamen verwendet. Er ist jedoch für komplexe, mehrstufige CAD-Aufgaben bei weitem nicht ausreichend, da er den dynamischen Zustand des Modells ignoriert.

3.2. Fortgeschrittenes Context Engineering: Aufbau eines dynamischen "Bewusstseins"

Context Engineering ist die "delikate Kunst und Wissenschaft, das Kontextfenster mit genau den richtigen Informationen für den nächsten Schritt zu füllen".[29] Es geht über den einfachen Abruf statischer Dokumente hinaus und betrachtet den Kontext als ein dynamisches, orchestriertes Informations-Ökosystem, das das Verhalten des Agenten steuert. Für einen CAD-Agenten ist dieser Ansatz von entscheidender Bedeutung, da seine Aktionen stark vom aktuellen Zustand des 3D-Modells abhängen.

Ein effektiv gestalteter Kontext für einen FreeCAD-Agenten würde aus mehreren

dynamisch zusammengestellten Komponenten bestehen, die bei jedem Inferenzschritt neu bewertet werden [17]:

- **System-Prompt & Stabiler Präfix:** Ein sorgfältig formulierter System-Prompt, der die Rolle des Agenten ("Du bist ein Experte für FreeCAD-Python-Scripting"), seine Ziele (z. B. "Erzeuge immer Code, der zu vollständig bestimmten Skizzen führt"), Einschränkungen und das gewünschte Ausgabeformat definiert. Der vordere Teil dieses Prompts sollte so stabil wie möglich gehalten werden, um die Trefferquote des KV-Caches zu maximieren, was die Leistung verbessert und die Inferenzkosten senkt.[17]
- **Aufgabe & Plan:** Das übergeordnete Ziel und der aktuelle Schritt aus dem Plan, der vom Planer-Agenten (siehe Abschnitt 4) erstellt wurde. Das ständige Wiederholen des Plans am Ende des Kontexts ("Recitation") hilft dem Agenten, den Fokus zu behalten und das "Lost-in-the-Middle"-Problem zu mildern, indem das globale Ziel in der jüngsten Aufmerksamkeit des Modells gehalten wird.[17]
- **Dynamische API-Schnipsel (RAG):** Nicht die gesamte API-Dokumentation, sondern nur die hochrelevanten Ausschnitte, die für den *aktuellen* Unterschritt benötigt werden.
- **Aktueller Dokumentenzustand:** Dies ist die kritischste und dynamischste Komponente. Es ist keine vollständige Serialisierung der .FCStd-Datei, sondern eine token-effiziente, strukturierte Zusammenfassung der relevanten Aspekte der aktuellen 3D-Umgebung. Dies könnte beinhalten:
    - Eine Liste der vorhandenen Objekte mit ihren Namen, Typen und wichtigen Eigenschaften (z. B. {'name': 'MyBox', 'type': 'Part::Box', 'position': (0,0,0)}).
    - Eine Zusammenfassung der Topologie des aktiven Objekts, möglicherweise als vereinfachte Graph-Darstellung (siehe Abschnitt 5).
    - Die Liste der Geometrien und Zwangsbedingungen in der aktuell bearbeiteten Skizze.
- **Konversations- & Aktionshistorie:** Eine Zusammenfassung der letzten Interaktionen mit dem Benutzer und der eigenen Aktionen des Agenten sowie deren Ergebnisse (Beobachtungen). Dies schließt nicht nur erfolgreiche Aktionen ein, sondern, was entscheidend ist, auch fehlgeschlagene Aktionen und die zugehörigen Fehlermeldungen oder Stack-Traces. Das explizite Einbeziehen von Fehlern ermöglicht es dem LLM, aus seinen Fehlern zu lernen und den nächsten Versuch als ein Debugging-Problem zu formulieren.[17]

Die Zusammenstellung dieser Komponenten ist kein einmaliger Abruf, sondern ein kontinuierlicher Prozess der Aktualisierung eines strukturierten "Zustandsobjekts", das bei jedem Schritt an das LLM übergeben wird. Dieses Zustandsobjekt *ist* die

Wahrnehmung des Agenten, sein gesamtes Sichtfeld auf die Aufgabe.

3.3. Verwaltung des Kontextfensters

CAD-Projekte sind von Natur aus komplex und erfordern viele aufeinanderfolgende Schritte. Selbst bei sorgfältigem Context Engineering kann die Länge des Kontexts schnell die Grenzen des LLM-Kontextfensters überschreiten. Darüber hinaus neigt die Leistung von LLMs dazu, auch innerhalb der technischen Grenzen abzunehmen, je länger der Kontext wird.[17] Daher sind Strategien zur Komprimierung und Verwaltung des Kontexts unerlässlich.

- **Zusammenfassung:** Ältere Teile der Konversationshistorie oder der Aktionshistorie können durch einen sekundären LLM-Aufruf zusammengefasst werden. Anstatt einer wortgetreuen Aufzeichnung der letzten 20 Schritte könnte der Kontext eine Zusammenfassung enthalten wie: "Zusammenfassung der vorherigen Schritte: Eine Grundplatte wurde erfolgreich erstellt und vier Löcher wurden hinzugefügt. Der letzte Versuch, eine Verrundung hinzuzufügen, schlug mit einem Topologiefehler fehl."
- **Wiederherstellbare Komprimierung (Restorable Compression):** Diese Technik besteht darin, detaillierte Informationen aus dem Kontext zu entfernen, aber einen "Zeiger" darauf zu behalten. Anstatt beispielsweise die vollständige Liste der Vertices und Kanten eines komplexen Teils in den Kontext aufzunehmen, würde man nur seinen Namen und einen Hinweis darauf aufnehmen, dass seine detaillierte Geometrie bei Bedarf über die API abgefragt werden kann.[17] Dies reduziert die Token-Anzahl drastisch, ohne Informationen dauerhaft zu verlieren.
- **Strukturierte Repräsentation:** Die Verwendung von strukturierten Formaten wie JSON oder YAML für die Darstellung des Dokumentenzustands ist wesentlich token-effizienter als eine Beschreibung in natürlicher Sprache.

Effektives Context Engineering ist somit eine Form des "kognitiven Scaffoldings" für das LLM. Anstatt das Modell zu zwingen, aus den ersten Prinzipien über eine Domäne zu schlussfolgern, für die es keine angeborene Intuition hat, baut die Architektur ein Gerüst aus vorverarbeiteten, strukturierten und hochrelevanten Informationen. Wir konstruieren das Arbeitsgedächtnis und den Aufmerksamkeitsfokus der KI, um ein ansonsten unlösbares Problem in eine Reihe von handhabbaren, kontextualisierten Unterschritten zu zerlegen. Dieser Ansatz nutzt die Stärken des LLM bei der Mustererkennung und Sprachverarbeitung innerhalb eines eng definierten, gut

vorbereiteten Kontexts und vermeidet seine Schwächen bei der präzisen, zustandsbehafteten und langfristigen Planung.

Abschnitt 4: Eine elegante Lösung: Eine Multi-Agenten-, Solver-gestützte Architektur

Die Synthese der vorangegangenen Analysen – die Eigenheiten des FreeCAD-Ökosystems, die inhärenten Schwächen von LLMs im geometrischen Bereich und die Notwendigkeit eines hochentwickelten Kontexts – mündet in einem architektonischen Vorschlag, der auf funktionaler Dekomposition und der Zusammenarbeit spezialisierter Agenten basiert. Die "Eleganz" dieser Lösung liegt nicht in der Komplexität eines einzelnen, monolithischen KI-Modells, sondern in der intelligenten Aufteilung des Gesamtproblems in überschaubare Teilaufgaben, die jeweils von einem dafür optimierten Agenten oder Werkzeug bearbeitet werden. Dieser Ansatz wird durch moderne Agenten-Entwicklungsframeworks wie LangGraph unterstützt [30] und folgt dem zentralen Prinzip, das geometrische Schließen explizit vom LLM an den robusten, nativen Constraint-Solver von FreeCAD auszulagern – eine Schlüsselstrategie, die in der aktuellen Forschung an der Schnittstelle von KI und CAD an Bedeutung gewinnt.[14]

4.1. Architektonische Übersicht: Das ReAct-Paradigma

Das gesamte System operiert nach einem **ReAct (Reason, Act)**-Zyklus, einem bewährten Paradigma für autonome Agenten.[18] In jeder Iteration dieses Zyklus durchläuft das Agentensystem die folgenden Phasen:

1. **Reason (Schlussfolgern):** Der Agent analysiert den aktuellen Zustand (basierend auf dem konstruierten Kontext) und das übergeordnete Ziel. Er formuliert einen Gedanken oder eine Hypothese über den nächsten notwendigen Schritt.
2. **Act (Handeln):** Basierend auf der Schlussfolgerung wählt der Agent ein verfügbares Werkzeug (eine "Aktion") aus und legt dessen Parameter fest. In unserem Fall wäre eine Aktion beispielsweise "rufe den Coder-Agenten auf, um eine Skizze zu erstellen" oder "stelle dem Benutzer eine klärende Frage".
3. **Observe (Beobachten):** Die Aktion wird ausgeführt, und das Ergebnis wird

beobachtet. Dies kann die erfolgreiche Erstellung eines neuen Objekts, eine Fehlermeldung aus dem FreeCAD-Kernel oder eine Antwort vom Benutzer sein. Diese Beobachtung wird dem Kontext für die nächste "Reason"-Phase hinzugefügt.

Dieser iterative Prozess ermöglicht es dem System, komplexe, mehrstufige Probleme zu lösen, auf unerwartete Ergebnisse zu reagieren und aus Fehlern zu lernen, anstatt einem starren, vordefinierten Skript zu folgen.

4.2. Der Planer-Agent: Der strategische Verstand

Der Planer-Agent agiert auf der höchsten Abstraktionsebene und bildet die Schnittstelle zur menschlichen Absicht. Seine primäre Aufgabe ist es, die oft vagen und mehrdeutigen Anweisungen des Benutzers in einen strukturierten, maschinenlesbaren Plan zu übersetzen.

- **Rolle:** Interpretation von hochrangigen Benutzerzielen, wie z. B. "Erstelle einen 100mm L-Winkel mit vier Befestigungslöchern und einer Verstärkungsrippe".
- **Prozess:**
  1. **Anforderungsanalyse:** Ähnlich dem in der Forschung beschriebenen CADDesigner-Agenten [18], führt der Planer einen interaktiven Dialog mit dem Benutzer, um Unklarheiten zu beseitigen und fehlende Informationen zu sammeln. Er könnte fragen: "Welchen Durchmesser sollen die Befestigungslöcher haben?", "Sollen die Löcher gesenkt werden?" oder "Wo soll die Rippe platziert werden?".
  2. **Aufgabendekomposition:** Sobald die Anforderungen ausreichend spezifiziert sind, zerlegt der Planer das hochrangige Ziel in eine geordnete Sequenz von logischen, atomaren Teilzielen. Für den L-Winkel könnte der Plan wie folgt aussehen: ``.
  3. **Chain-of-Thought (CoT) Reasoning:** Der Agent nutzt CoT-Techniken, um seinen Denkprozess zu externalisieren.[33] Der generierte Plan wird nicht nur intern verwendet, sondern wird zu einem expliziten Teil des Kontexts für die nachfolgenden Agenten und kann dem Benutzer zur Überprüfung und Bestätigung vorgelegt werden. Dies erhöht die Transparenz und Steuerbarkeit des gesamten Prozesses.

4.3. Der Coder-Agent: Der API-Spezialist

Der Coder-Agent ist ein hochspezialisierter "Arbeiter", dessen einzige Aufgabe es ist, ein einzelnes, atomares Teilziel aus dem Plan des Planer-Agenten in ausführbaren FreeCAD-Python-Code zu übersetzen.

- **Rolle:** Übersetzung eines einzelnen, klar definierten Teilziels (z. B. "Erstelle Skizze auf Grundplatte für Befestigungslöcher") in syntaktisch korrekten Python-Code.
- **Kontext:** Sein Kontext ist für jede Aufgabe eng fokussiert und wird dynamisch zusammengestellt. Er enthält nur das, was für die unmittelbare Aufgabe relevant ist: das spezifische Teilziel, die per RAG abgerufene API-Dokumentation für die benötigten Funktionen (z. B. addObject('Sketcher::SketchObject'), addGeometry, addConstraint) und eine Zusammenfassung des relevanten geometrischen Zustands (z. B. die Namen und Typen der verfügbaren Flächen, an die die Skizze angehängt werden kann).
- **Fine-Tuning:** Im Gegensatz zum Planer-Agenten, der ein großes, allgemeines LLM sein kann, wäre der Coder-Agent idealerweise ein kleineres, effizienteres Modell, das speziell auf die FreeCAD-API feinabgestimmt wurde. Dies macht ihn zu einem Experten für die spezifische Syntax, die Idiome und die häufigen Muster der FreeCAD-Skripterstellung, was zu qualitativ hochwertigerem und zuverlässigerem Code führt.

4.4. Der Verifikations- & Verfeinerungs-Agent: Die Qualitätssicherungs-Engine

Dieser Agent bildet die entscheidende Feedbackschleife, die die generativen Fähigkeiten des Coder-Agenten in der deterministischen Realität des CAD-Systems verankert. Er ist verantwortlich für die Ausführung, Validierung und Bereitstellung von korrektivem Feedback und fungiert als Brücke zwischen der probabilistischen KI und der exakten Welt der Geometrie.

- **Rolle:** Ausführung des generierten Codes, Überprüfung auf syntaktische und geometrische Korrektheit und Erstellung von strukturiertem Feedback.
- **Prozess:**
    1. **Sandboxed Execution:** Der vom Coder-Agenten generierte Python-Code wird in einer isolierten Umgebung ausgeführt, z. B. in einer Headless-Instanz von FreeCAD, um unbeabsichtigte Änderungen am Hauptdokument des

Benutzers zu verhindern.

2. **Symbolische Verifikation:** Der Agent fängt alle Python-Laufzeitfehler ab (z. B. SyntaxError, TypeError, AttributeError). Der vollständige Stack-Trace wird erfasst, um eine detaillierte Fehleranalyse zu ermöglichen.
3. **Geometrische Validierung:** Dies ist der wichtigste Schritt. Nach erfolgreicher Code-Ausführung führt der Agent eine Reihe von Überprüfungen am resultierenden FreeCAD-Dokument durch:
   - **Solver-Status:** War die Skizzenerstellung erfolgreich? Dies kann durch Abfrage von Eigenschaften wie sketch.HasError überprüft werden.
   - **Geometrische Integrität:** Ist der resultierende Volumenkörper gültig und mannigfaltig? Funktionen wie shape.isNull() oder shape.isValid() aus der Part-API werden hierfür verwendet.
   - **Plan-Einhaltung:** Wurde das Teilziel erreicht? Dies wird durch einen Vergleich des Dokumentenzustands vor und nach der Operation überprüft. Wenn das Ziel war, ein Loch zu erstellen, muss nach der Ausführung eine entsprechende topologische Änderung nachweisbar sein.
4. **Feedback-Generierung:** Die Ergebnisse werden in einem strukturierten Format (z. B. JSON) zusammengefasst, das als Beobachtung für den nächsten ReAct-Zyklus dient.
   - **Erfolg:** {'status': 'success', 'message': 'Ein neues Pad-Objekt "Pad001" wurde erstellt.', 'newState':...}
   - **Syntaktischer Fehler:** {'status': 'error', 'type': 'SyntaxError', 'message': '...', 'script': '...'}
   - **Geometrischer Fehler:** {'status': 'error', 'type': 'GeometricValidationError', 'message': 'Die Skizze ist überbestimmt.', 'script': '...'}
5. **Iterative Verfeinerungsschleife:** Dieses Feedback wird dem Kontext des Coder-Agenten hinzugefügt, der dann angewiesen wird, den Code zu "debuggen" und eine korrigierte Version zu erstellen. Diese Schleife, die an die Methoden von Frameworks wie CAD-Coder erinnert [33], wird fortgesetzt, bis der Code gültig ist oder eine maximale Anzahl von Versuchen erreicht ist, woraufhin das Problem an den Planer-Agenten zur Neuplanung eskaliert werden kann.[35]

4.5. Bootstrapping des Coder-Agenten: Synthetische Datengenerierung

Um die Datenknappheit [22] zu überwinden und den Coder-Agenten effektiv zu spezialisieren, wird ein Prozess zur Generierung synthetischer Daten eingesetzt. Anstatt auf seltene, reale Datensätze angewiesen zu sein, nutzen wir ein leistungsstarkes "Lehrer"-Modell (z. B. GPT-4o oder Claude 3 Opus), um einen großen, qualitativ hochwertigen Trainingsdatensatz für ein kleineres, effizienteres "Schüler"-Modell (den Coder-Agenten) zu erstellen.

- **Methodik (Self-Instruct & Evol-Instruct):** Dieser Ansatz adaptiert bewährte Methoden aus der Forschung zur synthetischen Codegenerierung.[37]
    1. **Seed-Set:** Ein menschlicher Experte erstellt manuell ein kleines, aber vielfältiges Set (ca. 50-100 Beispiele) von qualitativ hochwertigen Paaren aus Anweisung und FreeCAD-Skript.
    2. **Self-Instruct:** Das Lehrer-Modell erhält einige dieser Beispiele als Few-Shot-Prompt und wird angewiesen, Tausende neuer, ähnlicher Anweisungs-Skript-Paare zu generieren.
    3. **Evol-Instruct:** In einem zweiten Schritt wird das Lehrer-Modell angewiesen, bestehende Anweisungen iterativ zu verkomplizieren (z. B. "füge eine weitere Zwangsbedingung hinzu", "kombiniere zwei Operationen") und dann die entsprechend komplexeren Skripte zu generieren. Dies erhöht die Vielfalt und den Schwierigkeitsgrad des Datensatzes.
    4. **Verifikation als Qualitätsfilter:** Der entscheidende Schritt ist, dass jedes synthetisch generierte Skript automatisch vom Verifikations-Agenten validiert wird. Nur Skripte, die fehlerfrei ausgeführt werden und geometrisch gültige Ergebnisse liefern, werden in den endgültigen Feinabstimmungsdatensatz aufgenommen. Dies stellt sicher, dass der Coder-Agent nicht nur lernt, syntaktisch korrekten, sondern auch funktional richtigen Code zu schreiben.

Diese Multi-Agenten-Architektur stellt eine explizite Anerkennung der "gezackten technologischen Grenze" der KI-Fähigkeiten dar.[39] Anstatt zu versuchen, ein einziges Modell für eine komplexe End-to-End-Aufgabe zu verwenden, für die es ungeeignet ist, zerlegen wir die Aufgabe in Teilprobleme und weisen jedes einer spezialisierten Komponente zu, die darin hervorragend ist. Die Verfeinerungsschleife verwandelt die Codegenerierung zudem konzeptionell in ein Problem des bestärkenden Lernens, bei dem das Feedback des Verifikations-Agenten als "Belohnungssignal" dient, das die Politik des Coder-Agenten zur Generierung besserer Skripte im Laufe der Zeit anleitet.

Agent	Primäre Rolle	Eingaben	Kernverarbeitung	Ausgaben	Schlüsseltechnologien
Planer-Age	Strategische	Mehrdeutige	Anforderung	Eine	LLM (z.B.

nt	s Schließen & Aufgabende komposition	r Benutzer-Prompt (Text, Skizze); Konversationshistorie; Semantisches Gedächtnis (Zusammenfassung).	sanalyse; Dialogmanagement; Chain-of-Thought-Reasoning zur Zerlegung von Zielen.	strukturierte, geordnete Liste von atomaren Teilzielen; Klärende Fragen an den Benutzer.	GPT-4o), ReAct-Loop.
Coder-Agent	Codegenerierung	Ein einzelnes atomares Teilziel; Konstruierter Kontext (RAG API-Schnipsel, aktueller Dokumentenzustand); Feedback vom Verifikations-Agenten.	Übersetzt Teilziel in FreeCAD-Python-Skript.	Ausführbarer Python-Code-String.	Feinabgestimmtes SLM, RAG, Context Engineering.
Verifikations- & Verfeinerungs-Agent	Ausführung & Qualitätssicherung	Python-Code vom Coder-Agenten; Dokumentenzustand vor der Ausführung.	Führt Code in einer Sandbox aus; Führt symbolische (Syntax) und geometrische (Solver, Mannigfaltigkeit) Validierungen durch.	Strukturiertes Feedback (Erfolg, Fehlermeldung, Bericht über geometrische Ungültigkeit); Dokumentenzustand nach der Ausführung.	Headless FreeCAD-Instanz, Python try-except, FreeCAD API-Prüfungen.

Abschnitt 5: Der "Geist" des Agenten: Langzeitgedächtnis und

Modellierung der Designabsicht

Um von einem reinen Werkzeug zu einem echten kollaborativen Partner zu werden, muss ein KI-Agent die Fähigkeit besitzen, über einzelne Sitzungen hinweg zu lernen, sich anzupassen und ein tiefes Verständnis für das Projekt und den Benutzer zu entwickeln. Ein einfaches RAG-System liefert zwar Wissen über die API, aber erst eine dedizierte Gedächtnisarchitektur ermöglicht Weisheit, Personalisierung und die Erfassung der schwer fassbaren "Designabsicht". Dieser Abschnitt beschreibt eine hybride Gedächtnisarchitektur, die von Modellen der menschlichen Kognition inspiriert ist [40] und speziell auf die Domäne des parametrischen CAD zugeschnitten ist.

5.1. Eine hybride Gedächtnisarchitektur

Das Gedächtnis des Agenten ist in zwei Hauptkomponenten unterteilt, die unterschiedliche Zeiträume und Funktionen abdecken:

- **Kurzzeit-/Arbeitsgedächtnis:** Dies ist das in Abschnitt 3 beschriebene, dynamisch konstruierte Kontextfenster. Es enthält alle Informationen, die für die unmittelbar anstehende Aufgabe relevant sind, und wird bei jedem Schritt des ReAct-Zyklus neu aufgebaut. Es ist flüchtig und auf die aktuelle Aufgabe beschränkt.
- **Langzeitgedächtnis (Long-Term Memory, LTM):** Dies ist ein persistentes Speichersystem, das es dem Agenten ermöglicht, Informationen über Sitzungen, Projekte und Interaktionen hinweg zu speichern und abzurufen.[40] Das LTM ist die Grundlage für Lernen, Personalisierung und das Verständnis der Designentwicklung. Unsere vorgeschlagene LTM-Architektur besteht aus drei komplementären Subsystemen.

5.2. Episodisches Gedächtnis: Das "Was und Wann" des Designs

- **Funktion:** Das episodische Gedächtnis ist ein chronologisches, unveränderliches Protokoll aller Interaktionen und Operationen, die während eines Designprozesses stattgefunden haben. Es zeichnet die genaue Abfolge auf: Benutzeranfragen, die

Chain-of-Thought-Argumentation des Planer-Agenten, der generierte Code des Coder-Agenten und die detaillierten Ergebnisse des Verifikations-Agenten (sowohl Erfolge als auch Misserfolge).[40]

- **Implementierung:** Dies kann relativ einfach als eine Zeitreihendatenbank oder eine Sammlung von strukturierten Log-Dateien (z. B. im JSONL-Format) implementiert werden. Jede "Episode" ist ein vollständiger ReAct-Zyklus mit einem Zeitstempel.
- **Anwendungsfall:** Dieses Gedächtnis ist von unschätzbarem Wert für die Fehlersuche und die Nachvollziehbarkeit. Es ermöglicht eine vollständige "Wiedergabe" einer Designsitzung und beantwortet die Frage: "Welche genauen Schritte haben zu diesem Ergebnis geführt?". Es bildet die Rohdatenbasis, aus der die anderen Gedächtnissysteme lernen können.

5.3. Semantisches Gedächtnis: Das "Warum und Wie" des Modells

- **Funktion:** Während das episodische Gedächtnis die lineare Geschichte aufzeichnet, speichert das semantische Gedächtnis das strukturierte, relationale Wissen über das CAD-Modell selbst, seine internen Abhängigkeiten und, was am wichtigsten ist, die *Begründung* für wichtige Designentscheidungen. Hier findet das eigentliche "Verständnis" des Modells statt.[40]
- **Implementierung: Ein B-Rep-Wissensgraph.** Die inhärente Graphenstruktur eines B-Rep-Modells ist die ideale Grundlage für das semantische Gedächtnis.[45]
    - **Knoten:** Repräsentieren topologische Entitäten (Flächen, Kanten, Ecken), geometrische Primitive (Ebenen, Zylinder), parametrische Merkmale (Pads, Pockets, Skizzen) und Zwangsbedingungen. Darüber hinaus werden auch abstraktere Konzepte wie "Designanforderung" oder "Begründung" als Knoten modelliert.
    - **Kanten:** Repräsentieren die vielfältigen Beziehungen innerhalb des Modells: ist_Teil_von (z. B. Fläche -> Volumenkörper), wird_begrenzt_durch (z. B. Fläche -> Schleife von Kanten), ist_bestimmt_durch (z. B. Kante -> Längen-Constraint).
- **Erfassung der Designabsicht:** Die entscheidende Neuerung ist die Einführung einer gerechtfertigt_durch-Kante. Wenn der Planer-Agent eine Entscheidung trifft (z. B. den Durchmesser eines Lochs auf 5 mm festlegt), kann er diese Entscheidung mit einer Begründung verknüpfen, die entweder aus der Benutzeranfrage abgeleitet wurde ("...für eine M5-Schraube") oder explizit erfragt wurde. Diese Begründung wird als eigener Knoten im Graphen gespeichert und

über die gerechtfertigt_durch-Kante mit dem entsprechenden Constraint-Knoten verbunden. Dies erfasst explizit die Designabsicht (Design Intent) und geht über die reine Geometrie hinaus.[19] Eine lineare Befehlshistorie ist unzureichend, um dieses komplexe Beziehungsgeflecht abzubilden. Ein Wissensgraph ermöglicht es dem Agenten, komplexe relationale Abfragen zu beantworten, wie z. B. "Welche Merkmale hängen von dieser Skizze ab?" oder "Warum wurde dieser Radius gewählt?". Dies liefert einen weitaus reichhaltigeren Kontext für die Planung und Codegenerierung als jede andere Repräsentation.

5.4. Prozedurales Gedächtnis: Das Erlernen des "Stils" des Benutzers

- **Funktion:** Das prozedurale Gedächtnis speichert und ruft erlernte Fähigkeiten, wiederkehrende Arbeitsabläufe und benutzerspezifische Designmuster ab.[40] Es ermöglicht dem Agenten, mit der Zeit effizienter, proaktiver und persönlicher zu werden.
- **Implementierung:**
  - **Workflow-Extraktion:** Durch die Analyse des episodischen Gedächtnisses auf wiederkehrende Sequenzen von Operationen kann der Agent häufige Arbeitsabläufe lernen. Wenn ein Benutzer beispielsweise immer wieder eine Skizze erstellt, diese extrudiert und dann die oberen Kanten verrundet, kann der Agent dieses Muster als "Standard-Block-Erstellungs-Workflow" erkennen und speichern.
  - **Stil-Parametrisierung:** Der Agent kann die Präferenzen eines Benutzers lernen, wie z. B. typische Verrundungsradien, bevorzugte Freiräume für Bohrungen oder häufig verwendete Materialeigenschaften. Diese Parameter werden mit dem Benutzerprofil verknüpft.
- **Anwendungsfall:** Wenn der Agent mit einer neuen, aber bekannten Aufgabe konfrontiert wird (z. B. "Erstelle einen weiteren Stützblock"), kann er sein prozedurales Gedächtnis abfragen: "Habe ich dieses Problem schon einmal gelöst? Was war der erfolgreiche Workflow?". Dies ermöglicht es ihm, nicht nur den nächsten Schritt, sondern ganze mehrstufige Lösungen vorzuschlagen und sich so von einem reaktiven Werkzeug zu einem proaktiven "Co-Piloten" zu entwickeln.[50]

Diese mehrschichtige Gedächtnisarchitektur ermöglicht eine entscheidende Entwicklung: den Übergang von einem reinen "Befehlsausführer" zu einem echten "Designpartner". Ohne LTM ist der Agent zustandslos und behandelt jede Anfrage als

ein isoliertes Problem. Mit einem LTM baut er eine Beziehung zum Benutzer und zum Projekt auf. Ein zustandsloser Agent kann den Befehl "mache ein 5-mm-Loch" ausführen. Ein Agent mit semantischem Gedächtnis kann die Anweisung "mache ein Loch für eine M5-Schraube" verstehen, indem er auf eine Wissensdatenbank über Schraubenfreiräume zugreift, ein 5,5-mm-Loch erstellt und dies mit der Begründung "Freiraum für M5" verknüpft. Ein Agent mit prozeduralem Gedächtnis, der beobachtet hat, wie der Benutzer mehrere solcher Löcher platziert, kann ein Muster erkennen und proaktiv fragen: "Möchten Sie, dass ich an den anderen drei Ecken dieser Platte nach demselben Muster M5-Freiraumlöcher hinzufüge?". Diese Progression, die durch die verschiedenen Gedächtnisschichten ermöglicht wird, hebt den Agenten in der Wertschöpfungskette von einem einfachen Werkzeug zu einem proaktiven, intelligenten Mitarbeiter.[43]

Abschnitt 6: Die Mensch-KI-Partnerschaft: Interaktion, Vertrauen und Erklärbarkeit

Die technologisch fortschrittlichste KI ist nutzlos, wenn der Endbenutzer ihr nicht vertraut oder nicht effektiv mit ihr interagieren kann. Insbesondere in einem sicherheitskritischen und präzisionsorientierten Bereich wie dem Ingenieurwesen ist der Aufbau von Vertrauen keine optionale Zusatzfunktion, sondern eine grundlegende Anforderung. Dieser letzte Abschnitt befasst sich mit den entscheidenden Aspekten der Mensch-Computer-Interaktion (Human-Computer Interaction, HCI), die sicherstellen, dass das vorgeschlagene System nicht als undurchsichtige "Black Box" agiert, sondern als transparenter, zuverlässiger und kollaborativer Partner im Designprozess.

6.1. Jenseits des Chatbots: Fortgeschrittene Interaktionsparadigmen

Eine einfache textbasierte Eingabeaufforderung ist zwar ein guter Ausgangspunkt, schöpft aber das Potenzial der Mensch-KI-Kollaboration bei weitem nicht aus. Eine fortschrittliche Architektur sollte flexiblere und leistungsfähigere Interaktionsmodelle unterstützen.

- **Mixed-Initiative-Interaktion:** Das System sollte einen fließenden Dialog

ermöglichen, bei dem sowohl der Mensch als auch die KI die Initiative ergreifen können.[51] Der Benutzer kann Befehle erteilen, aber die KI kann auch proaktiv klärende Fragen stellen ("Soll diese Kante ebenfalls verrundet werden?"), basierend auf ihrem prozeduralen Gedächtnis nächste Schritte vorschlagen ("Der nächste logische Schritt wäre, die Befestigungslöcher zu definieren. Soll ich damit beginnen?") oder auf potenzielle Designprobleme hinweisen.

- **Proaktive Designunterstützung:** Der Agent kann mit Domänenwissen ausgestattet werden, um den Benutzer aktiv zu unterstützen:
  - **Design for Manufacturability (DFM) Analyse:** Der Agent kann mit einem Regelwerk für verschiedene Fertigungsverfahren (z. B. 3D-Druck, CNC-Fräsen) ausgestattet werden. Während des Designprozesses kann er das Modell kontinuierlich analysieren und proaktiv warnen: "Achtung: Diese Wandstärke von 1 mm ist zu dünn für den FDM-3D-Druck mit einer 0,4-mm-Düse. Es wird empfohlen, sie auf mindestens 1,6 mm zu erhöhen".[52]
  - **Automatisierte Fehlerkorrektur:** Wenn der Verifikations-Agent einen behebbaren Fehler feststellt (z. B. eine leicht überbestimmte Skizze), könnte er dem Benutzer direkt eine Lösung vorschlagen: "Diese Skizze ist überbestimmt, da der Abstand sowohl horizontal als auch vertikal festgelegt ist. Soll ich die vertikale Abstandsbeschränkung entfernen, um das Problem zu beheben?".[55]
- **Multimodale Eingaben:** Während der anfängliche Fokus auf Texteingaben liegt, ist die Architektur so konzipiert, dass sie erweiterbar ist, um auch andere Eingabemodalitäten zu akzeptieren. Eine vom Benutzer gezeichnete Freihandskizze oder ein Foto eines bestehenden Teils könnte als Eingabe für den Planer-Agenten dienen, der diese visuellen Informationen dann in abstrakte Designziele übersetzt.[18]

6.2. Vertrauensbildung in einer sicherheitskritischen Domäne

Im Ingenieurwesen können Designfehler kostspielige oder sogar gefährliche Folgen haben. Vertrauen in ein KI-Werkzeug ist daher nicht verhandelbar und muss durch Designprinzipien aktiv gefördert werden. Vertrauen ist eine Funktion von Vorhersehbarkeit, Transparenz und Kontrolle.

- **Transparenz und Erklärbarkeit (Explainable AI, XAI):** Ein Benutzer wird einem System, dessen Entscheidungen er nicht nachvollziehen kann, niemals voll vertrauen.[58]

    - **Quellen zitieren:** Wenn der Agent Informationen liefert (z. B. über eine API-Funktion), sollte das RAG-System immer die genaue Quelle aus der Dokumentation zitieren, ähnlich wie es der Onshape AI Advisor tut.[61]
    - **Den Plan offenlegen:** Die Chain-of-Thought-Argumentation des Planer-Agenten sollte für den Benutzer sichtbar sein. Bevor der Agent mit der Ausführung eines mehrstufigen Plans beginnt, sollte der Benutzer die Möglichkeit haben, diesen zu überprüfen, zu bearbeiten und zu genehmigen.
    - **Aktionen mit Absicht verknüpfen:** Jedes von der KI erstellte Merkmal im Modell sollte im semantischen Gedächtnis mit der ursprünglichen Benutzeranfrage und der abgeleiteten Begründung verknüpft sein. Der Benutzer muss in der Lage sein, auf ein beliebiges Merkmal zu klicken und zu fragen: "Warum ist das hier?", und eine klare, nachvollziehbare Antwort zu erhalten (z. B. "Dieses Loch wurde mit einem Durchmesser von 5,5 mm erstellt, um den Freiraum für eine M5-Schraube gemäß Ihrer Anweisung zu gewährleisten.").
- **Verifikation und Benutzerkontrolle:** Der Mensch muss immer die letzte Instanz sein. Die Aktionen der KI sollten als Vorschläge präsentiert werden, die der Benutzer annehmen, ablehnen oder modifizieren kann. Die iterative Verfeinerungsschleife des Verifikations-Agenten sollte transparent sein und dem Benutzer zeigen, wie der Agent seine eigenen Fehler korrigiert, anstatt sie zu verbergen.[62] Dies gibt dem Benutzer die Gewissheit, dass ein rigoroser Qualitätssicherungsprozess stattfindet.

6.3. Auswirkungen auf die Fähigkeiten von Designern und geistiges Eigentum

Die Einführung eines leistungsstarken KI-Assistenten wirft wichtige Fragen über die Zukunft der Designberufe und die Urheberschaft von KI-generierten Werken auf.

- **Augmentation, nicht Substitution:** Das System ist bewusst als "Co-Pilot" konzipiert, der die Fähigkeiten des Designers erweitert, anstatt sie zu ersetzen.[50] Es automatisiert mühsame und repetitive Aufgaben (z. B. das Erstellen von Mustern, das Anwenden konsistenter Verrundungen, das Schreiben von Boilerplate-Code), um dem Designer Zeit für höherwertige, kreative und strategische Problemlösungen zu verschaffen. Die Forschung zu KI-Assistenten zeigt, dass eine zu passive Rolle des Menschen zu einem Abbau kognitiver Fähigkeiten und zu übermäßigem Vertrauen führen kann.[65] Ein Mixed-Initiative-System hält den Benutzer engagiert und fördert das kritische

Denken.

- **Geistiges Eigentum und Urheberschaft:** Die aktuelle Rechtslage bezüglich KI-generierter Werke ist komplex und in vielen Rechtsordnungen noch nicht gefestigt. Ein zentrales Prinzip, insbesondere im US-Recht, ist die Anforderung der menschlichen Urheberschaft.[66] Ein Werk, das vollständig autonom von einer KI erstellt wird, ist möglicherweise nicht urheberrechtlich schützbar. Unsere vorgeschlagene Architektur stärkt den Anspruch des menschlichen Benutzers auf die Urheberschaft. Da der Mensch die ursprüngliche, kreative Absicht liefert, den Plan der KI validiert und die endgültigen Designentscheidungen trifft, agiert die KI als ein außergewöhnlich fortschrittliches Werkzeug, aber die kreative Kontrolle und Autorität verbleibt beim Menschen.[68] Dieser "Human-in-the-Loop"-Ansatz ist somit nicht nur ein HCI-Prinzip, sondern auch eine rechtliche und kognitive Notwendigkeit.

Die folgende Tabelle vergleicht den vorgeschlagenen Ansatz mit den KI-Funktionen in etablierten kommerziellen CAD-Systemen, um die Neuheit und den strategischen Vorteil der vorgeschlagenen Architektur zu verdeutlichen.

Merkmal	Autodesk Fusion 360 [70]	SolidWorks [72]	Onshape [61]	Vorgeschlagenes FreeCAD Addon
Kernparadigma	Generatives Design (Optimierung), KI-Co-Pilot	Design-Assistent (Aufgabenautomatisierung)	AI Advisor (Fragen & Antworten zur Dokumentation)	Mixed-Initiative Designpartner
Eingabemodalität	Designparameter, Zwangsbedingungen	Benutzeraktionen (Klicks, Auswahlen)	Fragen in natürlicher Sprache	Natürliche Sprache, Skizzen (erweiterbar)
Primärer Anwendungsfall	Topologieoptimierung, Teilekonsolidierung	Automatisierung repetitiver Aufgaben (Verknüpfungen, Auswahl, Skizzenmuster)	Beantwortung von "Wie-mache-ich"-Fragen, Bereitstellung von Dokumentationslinks	End-to-End-Konzeptdesign aus mehrdeutiger Absicht

Codegenerierung	Nein (generiert Geometrie)	Nein (automatisiert UI-Aktionen)	Nein (liefert Code-Schnipsel für Featurescript)	Ja (Kernfunktion, generiert ausführbares Python)
Verifikationsschleife	Simulationsbasierte Validierung der generierten Optionen	N/A	N/A	Ja (Symbolische + Geometrische Validierung mit iterativer Verfeinerung)
Anpassungsfähigkeit	N/A	Lernt aus Benutzer-Workflows, um Vorschläge zu verbessern	N/A	Ja (Langzeitgedächtnis für Benutzerstil und Workflows)
Erklärbarkeit	Begrenzt (zeigt Diagramme, filtert Ergebnisse)	Begrenzt (Aktionen sind direkt)	Hoch (zitiert Quellen)	Hoch (legt Plan offen, verknüpft Aktionen mit Absicht, zitiert Quellen)

Schlussfolgerungen

Die Integration von künstlicher Intelligenz in die parametrische CAD-Modellierung stellt eine der vielversprechendsten, aber auch anspruchsvollsten Grenzen der modernen Softwareentwicklung dar. Die vorliegende Analyse hat die fundamentalen Herausforderungen aufgedeckt, die sich aus der Kluft zwischen der probabilistischen Natur von LLMs und der deterministischen Präzision von CAD-Systemen ergeben. Ein naiver Ansatz, der lediglich auf eine direkte Text-zu-Code-Übersetzung abzielt, ist aufgrund der Unfähigkeit von LLMs zu präzisem räumlichen Schließen, der Komplexität der Zwangsbedingungslogik und der Notwendigkeit einer langfristigen, zustandsbehafteten Planung zum Scheitern verurteilt.

Als Antwort auf diese Herausforderungen wurde eine "elegante" Lösungsarchitektur vorgeschlagen, die auf drei Grundpfeilern ruht:

1. **Kognitive Lastenverteilung:** Die Architektur lagert bewusst die Aufgaben aus,

bei denen LLMs schwach sind. Das komplexe geometrische Schließen wird nicht der KI aufgebürdet, sondern dem hochoptimierten, deterministischen Constraint-Solver von FreeCAD überlassen. Die Rolle der KI wird von der eines "Problemlösers" zu der eines "Problemformulierers" neu definiert, der die menschliche Absicht in eine maschinenlesbare, lösbare Spezifikation übersetzt.

2. **Funktionale Dekomposition durch Multi-Agenten-Systeme:** Anstelle eines monolithischen Modells wird das komplexe Problem des Designs in spezialisierte Aufgaben zerlegt, die von kollaborativen Agenten (Planer, Coder, Verifikator) bearbeitet werden. Diese funktionale Trennung ermöglicht es, für jede Teilaufgabe das am besten geeignete Werkzeug – sei es ein großes, kreatives LLM für die Planung oder ein kleines, feinabgestimmtes Modell für die Codierung – einzusetzen und schafft durch die explizite Verifikationsschleife eine robuste Fehlerkorrektur.
3. **Kontext als dynamisches Bewusstsein:** Der Erfolg des Systems hängt von einem fortschrittlichen Context Engineering ab, das über einfaches RAG hinausgeht. Durch die Konstruktion eines dynamischen Kontexts, der den Plan, den aktuellen Modellzustand und die Aktionshistorie umfasst, und die Implementierung einer hybriden Langzeitgedächtnisarchitektur kann der Agent von einem reinen Werkzeug zu einem echten Designpartner heranreifen, der lernt, sich anpasst und die Absichten des Benutzers versteht.

Die vorgeschlagene Architektur des "Cognitive Scaffolding" ist mehr als nur ein technischer Entwurf. Sie ist ein strategischer Rahmen, der die Stärken von Mensch und Maschine synergetisch kombiniert. Indem sie die KI als Co-Piloten positioniert, der mühsame prozedurale Aufgaben automatisiert, während der Mensch die kreative und strategische Kontrolle behält, schafft sie einen Weg für die Einführung von KI im Ingenieurwesen, der Vertrauen, Transparenz und die Augmentation menschlicher Fähigkeiten in den Vordergrund stellt. Die Umsetzung dieses Rahmens innerhalb des offenen und erweiterbaren FreeCAD-Ökosystems bietet die einzigartige Gelegenheit, die nächste Generation intelligenter Designwerkzeuge zu gestalten und die Grenzen dessen, was in der computergestützten Konstruktion möglich ist, neu zu definieren.

Referenzen

1. Your own 3D parametric modeler - FreeCAD, Zugriff am August 6, 2025, https://www.freecad.org/features.php
2. Official source code of FreeCAD, a free and opensource multiplatform 3D parametric modeler. - GitHub, Zugriff am August 6, 2025, https://github.com/FreeCAD/FreeCAD
3. FreeCAD-documentation/wiki/Part_scripting.md at main - GitHub, Zugriff am August 6, 2025,

https://github.com/FreeCAD/FreeCAD-documentation/blob/main/wiki/Part_scripting.md
4. A gentle introduction · A FreeCAD manual - Yorik van Havre, Zugriff am August 6, 2025, https://yorikvanhavre.gitbooks.io/a-freecad-manual/content/python_scripting/a_gentle_introduction.html
5. Introduction to Multi-Body Boolean Operations | Basic Beginners FreeCAD Lesson 33, Zugriff am August 6, 2025, https://www.youtube.com/watch?v=Z41Qx7453zk
6. Learn FreeCAD: #8 Boolean Operations - Tutorial - YouTube, Zugriff am August 6, 2025, https://www.youtube.com/watch?v=Y1gPG16mDZI
7. FreeCAD Sketch Tutorial - Part 2. Constraints Fundamentals, Construction Elements. Multiple Sketches - YouTube, Zugriff am August 6, 2025, https://www.youtube.com/watch?v=ZJiBzHsGaVI
8. FreeCAD-documentation/wiki/Sketcher_Workbench.md at main - GitHub, Zugriff am August 6, 2025, https://github.com/FreeCAD/FreeCAD-documentation/blob/main/wiki/Sketcher_Workbench.md
9. Adding or modifying constraints with Python (Become an expert) - FreeCAD [How-to] [Book], Zugriff am August 6, 2025, https://www.oreilly.com/library/view/freecad-how-to/9781849518864/ch01s11.html
10. How To Script A Sketch In FreeCAD Using Python - - SingerLinks, Zugriff am August 6, 2025, https://singerlinks.com/2023/11/how-to-script-a-sketch-in-freecad-using-python/
11. [2502.09819] A Solver-Aided Hierarchical Language for LLM-Driven CAD Design - arXiv, Zugriff am August 6, 2025, https://arxiv.org/abs/2502.09819
12. Advice for a beginner? : r/FreeCAD - Reddit, Zugriff am August 6, 2025, https://www.reddit.com/r/FreeCAD/comments/rwrtsf/advice_for_a_beginner/
13. Mastering Constraints in Parametric Design - Number Analytics, Zugriff am August 6, 2025, https://www.numberanalytics.com/blog/mastering-constraints-parametric-design
14. A Solver-Aided Hierarchical Language for LLM-Driven CAD Design - arXiv, Zugriff am August 6, 2025, https://arxiv.org/html/2502.09819v1
15. The Challenges of Creating Realistic 3D Objects using AI - CGI.Backgrounds, Zugriff am August 6, 2025, https://www.cgibackgrounds.com/blog/the-challenges-of-creating-realistic-3d-objects-using-ai
16. Parametric Design: Harnessing AI Computational Power, Zugriff am August 6, 2025, https://www.ai-scaleup.com/ai-for-architecture/parametric-design/
17. Context Engineering for AI Agents: Lessons from Building Manus, Zugriff am August 6, 2025, https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus

18. CADDesigner: Conceptual Design of CAD Models Based on General-Purpose Agent - arXiv, Zugriff am August 6, 2025, https://arxiv.org/html/2508.01031v2
19. Design Intent in CAD: A Comprehensive Guide - Number Analytics, Zugriff am August 6, 2025, https://www.numberanalytics.com/blog/design-intent-in-cad-comprehensive-guide
20. VITRUVION:AGENERATIVE MODEL OF PARAMETRIC CAD SKETCHES - OpenReview, Zugriff am August 6, 2025, https://openreview.net/pdf?id=Ow1C7s3UcY
21. Toward AI-driven Multimodal Interfaces for Industrial CAD Modeling - arXiv, Zugriff am August 6, 2025, https://arxiv.org/html/2503.16824v1
22. AI-Powered CAD Model Generators: Progress and Challenges | PixelDojo News, Zugriff am August 6, 2025, https://pixeldojo.ai/industry-news/ai-powered-cad-model-generators-progress-and-challenges
23. Large Language Models for Computer-Aided Design: A Survey - arXiv, Zugriff am August 6, 2025, https://arxiv.org/html/2505.08137v1
24. What is Retrieval-Augmented Generation (RAG)? - Google Cloud, Zugriff am August 6, 2025, https://cloud.google.com/use-cases/retrieval-augmented-generation
25. What is Retrieval Augmented Generation (RAG)? - Databricks, Zugriff am August 6, 2025, https://www.databricks.com/glossary/retrieval-augmented-generation-rag
26. What is RAG (Retrieval Augmented Generation)? - IBM, Zugriff am August 6, 2025, https://www.ibm.com/think/topics/retrieval-augmented-generation
27. FreeCAD source documentation, Zugriff am August 6, 2025, https://freecad.github.io/SourceDoc/
28. FreeCAD: Python Tutorials - YouTube, Zugriff am August 6, 2025, https://www.youtube.com/playlist?list=PLpzgvcyYOSTDGU5tYikH14x5JUoFFB0o7
29. A Comprehensive Guide to Context Engineering for AI Agents | by Tamanna - Medium, Zugriff am August 6, 2025, https://medium.com/@tam.tamanna18/a-comprehensive-guide-to-context-engineering-for-ai-agents-80c86e075fc1
30. Vertex AI Agent Builder | Google Cloud, Zugriff am August 6, 2025, https://cloud.google.com/products/agent-builder
31. A Solver-Aided Hierarchical Language For LLM-Driven CAD Design | OpenReview, Zugriff am August 6, 2025, https://openreview.net/forum?id=2qJXhflNbR
32. CADDesigner: Conceptual Design of CAD Models Based on General-Purpose Agent - arXiv, Zugriff am August 6, 2025, https://www.arxiv.org/abs/2508.01031
33. CAD-Coder: Text-to-CAD Generation with Chain-of-Thought and Geometric Reward | Request PDF - ResearchGate, Zugriff am August 6, 2025, https://www.researchgate.net/publication/392105124_CAD-Coder_Text-to-CAD_Generation_with_Chain-of-Thought_and_Geometric_Reward
34. [Literature Review] CAD-Coder: Text-to-CAD Generation with Chain-of-Thought and Geometric Reward - Moonlight, Zugriff am August 6, 2025,

https://www.themoonlight.io/en/review/cad-coder-text-to-cad-generation-with-chain-of-thought-and-geometric-reward
35. Adaptive RAG for CAD: Dynamic Object Generation with LlamaIndex & Vertex AI - Medium, Zugriff am August 6, 2025, https://medium.com/google-cloud/adaptive-rag-for-cad-dynamic-object-generation-with-llamaindex-vertex-ai-fd26d23ffcd5
36. Generating CAD Code with Vision-Language Models for 3D Designs - arXiv, Zugriff am August 6, 2025, https://arxiv.org/html/2410.05340v2
37. Synthetic Data Generation Using Large Language Models: Advances in Text and Code, Zugriff am August 6, 2025, https://arxiv.org/html/2503.14023v1
38. Synthetic Data Generation Using Large Language Models ... - arXiv, Zugriff am August 6, 2025, https://arxiv.org/pdf/2503.14023
39. Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality, Zugriff am August 6, 2025, https://www.hbs.edu/ris/Publication%20Files/24-013_d9b45b68-9e74-42d6-a1c6-c72fb70c7282.pdf
40. What Is AI Agent Memory? | IBM, Zugriff am August 6, 2025, https://www.ibm.com/think/topics/ai-agent-memory
41. (PDF) Memory Architectures in Long-Term AI Agents: Beyond Simple State Representation, Zugriff am August 6, 2025, https://www.researchgate.net/publication/388144017_Memory_Architectures_in_Long-Term_AI_Agents_Beyond_Simple_State_Representation
42. Introduction to FreeCAD Part 1: Getting Started | DigiKey - YouTube, Zugriff am August 6, 2025, https://m.youtube.com/watch?v=8VPYTTnqmfs&pp=0gcJCdgAo7VqN5tD
43. Long-Term Memory for AI Agents | by CortexFlow | The Software ..., Zugriff am August 6, 2025, https://medium.com/the-software-frontier/long-term-memory-for-ai-agents-1d93516c08ae
44. Inside the Mind of an AI Agent: Architectures, Memory Models, and Decision Loops, Zugriff am August 6, 2025, https://aijourn.com/inside-the-mind-of-an-ai-agent-architectures-memory-models-and-decision-loops/
45. Deep Learning with Solid Models - Karl D.D. Willis, Zugriff am August 6, 2025, https://www.karlddwillis.com/deep-learning-with-solid-models/
46. BRepNet: A topological message passing system for solid models - Autodesk Research, Zugriff am August 6, 2025, https://www.research.autodesk.com/publications/brep-net/
47. BRT: Boundary representation learning via Transformer - arXiv, Zugriff am August 6, 2025, https://arxiv.org/html/2504.07134v1
48. [Literature Review] Geometric Deep Learning for Computer-Aided Design: A Survey, Zugriff am August 6, 2025, https://www.themoonlight.io/en/review/geometric-deep-learning-for-computer-aided-design-a-survey

49. Learning to Design From Humans: Imitating Human Designers Through Deep Learning | J. Mech. Des. | ASME Digital Collection, Zugriff am August 6, 2025, https://asmedigitalcollection.asme.org/mechanicaldesign/article/141/11/111102/955339/Learning-to-Design-From-Humans-Imitating-Human
50. Generative AI and the Nature of Work - Harvard Business School, Zugriff am August 6, 2025, https://www.hbs.edu/ris/download.aspx?name=25-021.pdf
51. AI and the Future of Human-Computer Interaction | by Mike ..., Zugriff am August 6, 2025, https://medium.com/@mike.anderson007/ai-and-the-future-of-human-computer-interaction-87b74524d906
52. AI in DFM: Revolutionizing Design - Number Analytics, Zugriff am August 6, 2025, https://www.numberanalytics.com/blog/ai-in-dfm-revolutionizing-design
53. Streamlining DFM with CAD - Number Analytics, Zugriff am August 6, 2025, https://www.numberanalytics.com/blog/streamlining-dfm-with-cad
54. Design for Manufacturing Software | Autodesk, Zugriff am August 6, 2025, https://www.autodesk.com/solutions/design-for-manufacturing-software
55. Computer-Aided Design with AI | Autodesk, Zugriff am August 6, 2025, https://www.autodesk.com/solutions/computer-aided-design-with-ai
56. CADReview: Automatically Reviewing CAD Programs with Error Detection and Correction, Zugriff am August 6, 2025, https://arxiv.org/html/2505.22304v1
57. Free2CAD: parsing freehand drawings into CAD commands - ResearchGate, Zugriff am August 6, 2025, https://www.researchgate.net/publication/362206634_Free2CAD_parsing_freehand_drawings_into_CAD_commands
58. Building AI Systems That Prioritise User Safety, Trust & Protection | by Tejj - Prototypr, Zugriff am August 6, 2025, https://blog.prototypr.io/designing-ai-systems-that-prioritise-user-safety-trust-and-protection-1f5c88538779
59. AI Trust and Safety | Teradata, Zugriff am August 6, 2025, https://www.teradata.com/insights/ai-and-machine-learning/ai-and-machine-learning-ai-trust-safety
60. Artificial intelligence in safety-critical systems: a systematic review - Emerald Insight, Zugriff am August 6, 2025, https://www.emerald.com/insight/content/doi/10.1108/imds-07-2021-0419/full/html
61. AI for Better CAD Design - Onshape, Zugriff am August 6, 2025, https://www.onshape.com/en/features/ai-advisor
62. Building Trust in AI: A New Era of Human-Machine Teaming | Center for Security and Emerging Technology - CSET, Zugriff am August 6, 2025, https://cset.georgetown.edu/article/building-trust-in-ai-a-new-era-of-human-machine-teaming/
63. Building Trust in AI: Introducing i.AI's Assurance Principles - Blogs, Zugriff am August 6, 2025, https://ai.gov.uk/blogs/building-trust-in-ai-introducing-i-ai-s-assurance-principles

64. The Impact of Artificial Intelligence on Design: Enhancing Creativity and Efficiency, Zugriff am August 6, 2025, https://www.researchgate.net/publication/384953179_The_Impact_of_Artificial_Intelligence_on_Design_Enhancing_Creativity_and_Efficiency
65. Microsoft says AI tools such as Copilot or ChatGPT are affecting critical thinking at work – staff using the technology encounter 'long-term reliance and diminished independent problem-solving' - ITPro, Zugriff am August 6, 2025, https://www.itpro.com/technology/artificial-intelligence/ai-tools-critical-thinking-reliance
66. Copyright and Artificial Intelligence, Part 2 Copyrightability Report - U.S. Copyright Office, Zugriff am August 6, 2025, https://www.copyright.gov/ai/Copyright-and-Artificial-Intelligence-Part-2-Copyrightability-Report.pdf
67. What Is an "Author"?-Copyright Authorship of AI Art Through a Philosophical Lens | Published in Houston Law Review, Zugriff am August 6, 2025, https://houstonlawreview.org/article/92132-what-is-an-author-copyright-authorship-of-ai-art-through-a-philosophical-lens
68. AI generated designs: innovation or infringement - Waterfront Law, Zugriff am August 6, 2025, https://waterfront.law/ai-generated-designs-innovation-or-infringement/
69. Intellectual Property Rights and AI-Generated Content — Issues in Human Authorship, Fair Use Doctrine, and Output Liability | by Adnan Masood, PhD. | Medium, Zugriff am August 6, 2025, https://medium.com/@adnanmasood/intellectual-property-rights-and-ai-generated-content-issues-in-human-authorship-fair-use-8c7ec9d6fdc3
70. Generative Design for Manufacturing | Autodesk Fusion, Zugriff am August 6, 2025, https://www.autodesk.com/solutions/generative-design/manufacturing
71. Generative Design in Autodesk Fusion: Revolutionizing Design with AI - Fusion Blog, Zugriff am August 6, 2025, https://www.autodesk.com/products/fusion-360/blog/generative-design-in-autodesk-fusion-revolutionizing-design-with-ai/
72. Evolve Your Design Workflows with AI | SOLIDWORKS, Zugriff am August 6, 2025, https://www.solidworks.com/lp/evolve-your-design-workflows-ai
73. AI Cannot Replace Creativity but It Can Help - The SOLIDWORKS Blog -, Zugriff am August 6, 2025, https://blogs.solidworks.com/solidworksblog/2023/01/save-time-on-tedious-tasks-with-ai-features-in-3d-creator.html
74. Introducing Onshape AI Advisor - YouTube, Zugriff am August 6, 2025, https://www.youtube.com/watch?v=vpPQC9rWtYo