Agentic, Specification-Driven AI-Assisted CAD System for Mechanical Design

This project aims to develop an agentic, specification-driven AI-assisted CAD system that converts fully defined mechanical design inputs into valid, parametric 3D CAD models using established engineering principles rather than free-form shape generation. The system will accept structured natural-language or parameter-based prompts containing explicit design variables (for example, module, number of teeth, pressure angle, face width, and material for a spur gear), interpret these inputs into a machine-readable format, apply standardized mechanical design equations and constraints derived from classical machine design theory and relevant ISO/AGMA standards, and then programmatically construct the geometry using a parametric CAD kernel (such as OpenCascade via FreeCAD scripting or CadQuery). The initial implementation will target involute spur gears: the system will compute all fundamental dimensions (pitch circle, base circle, addendum, dedendum, tooth thickness, and involute profiles), generate accurate tooth geometry mathematically, pattern it to form the complete gear, and export fully editable CAD files (STEP/STL) suitable for downstream engineering use. The architecture will deliberately separate deterministic engineering logic from the AI interface, ensuring correctness, explainability, and reproducibility, while laying a scalable foundation for future extension to more complex mechanical components (such as shafts, gear assemblies, and IC engine parts) via additional rule sets, multi-agent coordination, and richer standards integration in later phases.

Core Features (Target State)

- **Specification-Driven Input**
  - Accepts structured natural language descriptions (e.g., “spur gear with module 2 mm, 20 teeth, 20° pressure angle, 10 mm face width, steel”).
  - Supports direct parameter-based input via forms or JSON (module, teeth, pressure angle, face width, material, etc.).
  - Validates inputs against basic engineering sanity checks (e.g., minimum tooth count for given module, non-negative dimensions).
- **Deterministic Engineering Logic Layer**
  - Implements standard gear design equations (pitch diameter, base diameter, addendum, dedendum, root diameter, circular pitch, tooth thickness).
  - Uses involute gear tooth theory to compute accurate 2D tooth profiles.
  - Encapsulates design logic in a clear, testable Python module separate from AI and CAD layers.
  - Provides design reports summarizing all computed parameters and assumptions.
- **Parametric CAD Generation**
  - Uses a Python-accessible CAD kernel (e.g., CadQuery / OpenCascade / FreeCAD Python API).
  - Builds fully parametric 3D models of involute spur gears from computed parameters.

    - Patterns a single mathematically-generated tooth around the pitch circle to form the full gear.
    - Exports industry-standard CAD formats:
        - STEP for parametric CAD workflows.
        - STL for rapid prototyping / 3D printing.
- **AI/LLM-Based Interface**
    - LLM-based natural language understanding to convert user requests into structured gear parameters.
    - Robust prompt design to enforce strict JSON/structured outputs.
    - Basic reasoning about defaults and missing values (e.g., assume 20° pressure angle and standard addendum/dedendum if unspecified).
    - Clear explanations of how the AI interpreted the input and what assumptions were made.
- **Multi-Agent System (Later Phases)**
    - Separate agents for:
        - Parsing and parameter extraction.
        - Engineering validation and calculations.
        - CAD model generation and export.
        - Design review and constraint checking.
    - Coordination logic for more complex components and assemblies (e.g., gear–shaft–bearing systems).
- **Extensibility to Other Components (Future)**
    - Ability to add new rule sets for shafts, gear pairs, gear trains, and IC engine components.
    - Modular architecture to plug in additional standards (ISO, AGMA) and domain-specific calculation modules.
    - Path toward generating assemblies and not just single parts.
- **Explainability and Reproducibility**
    - Clear logging of:
        - Input parameters and interpreted values.
        - Equations used and intermediate results.
        - Versioned design rules and standards applied.
    - Deterministic outputs for the same input (no random geometry).

Tech Stack (Initial and Planned)

- **Programming Language**
    - Python (primary language for AI, engineering logic, and CAD automation).
- **CAD and Geometry**
    - CadQuery (Pythonic parametric CAD on top of OpenCascade) for the MVP.
    - Optionally FreeCAD Python API or direct OpenCascade bindings in later stages.
- **AI / LLM Layer**
    - Hosted LLMs (e.g., OpenAI GPT-4 family) for early experimentation with natural language → parameters.
    - Optionally open-source models (e.g., Llama family) for local experimentation as skills grow.
    - LangChain / similar orchestration for prompt templates and structured outputs.
    - Later: agent frameworks such as CrewAI or AutoGen for multi-agent workflows.

- **Backend & APIs (for SaaS Stage)**
  - FastAPI (or similar) for REST API endpoints to:
    - Accept design requests.
    - Trigger AI + engineering + CAD pipeline.
    - Serve generated STEP/STL files.
  - Background task processing (e.g., Celery / RQ) for heavier CAD jobs (later).
- **Frontend (for SaaS Stage)**
  - Web UI using React/Next.js or a lightweight framework (Streamlit/Gradio for early demos).
  - 3D viewer integration (e.g., Three.js or a web viewer that can load STEP/STL).
- **Data & Knowledge**
  - Python modules encoding mechanical design rules (initially hard-coded).
  - Later: vector database (e.g., Chroma, Pinecone) + RAG for standards and references.
- **DevOps & Tooling (Later)**
  - Git/GitHub for version control.
  - Docker for containerization.
  - Cloud deployment (e.g., AWS/GCP/Azure) for SaaS version.

MVP Roadmap (Short and Focused)

Phase 1: Core Geometry and Engineering Logic (Spur Gear Only)

- Implement a **GearParameters** data class (module, teeth, pressure angle, face width, material).
- Implement a **SpurGearCalculator**:
  - Compute pitch diameter, base diameter, outer diameter, root diameter.
  - Compute circular pitch and tooth thickness.
  - Generate involute tooth profile points mathematically.
- Write unit tests to verify formula correctness for sample cases from textbooks.

Phase 2: CAD Model Generation

- Use CadQuery (or chosen CAD kernel) to:
  - Create a 2D tooth profile from involute points.
  - Extrude the profile to face width.
  - Pattern the tooth around the pitch circle to form a full gear.
  - Add a central bore/hub if needed.
- Implement export to **STEP** and **STL**.
- Validate output by loading models into a CAD program (FreeCAD, Fusion 360, etc.).

Phase 3: Simple AI Parameter Extraction

- Integrate an LLM (e.g., GPT-4o-mini) with:
  - A prompt to extract gear parameters from natural language into a strict JSON schema.
- Implement a small script:
  - Input: user text description.
  - Output: generated **gear.step** and **gear.stl** using the previously built pipeline.
- Add basic error handling and fallback defaults.

Phase 4: Wrap as a Minimal Application

- Create a simple CLI or minimal web interface:
  - User enters text or form parameters.
  - System runs the pipeline and returns file paths or download links (for local use initially).
- Document the full flow and assumptions.

Steps to Reach the Final Goal: End-to-End SaaS Product

1. Strengthen and Generalize the Engineering Core

- Refine and document the gear design equations and constraints.
- Add:
  - Strength calculations (e.g., Lewis/AGMA).
  - Clear constraint checks (e.g., undercut, minimum tooth counts, manufacturing limits).
- Design the core as a **library** with clean APIs so that AI and UI layers depend on it, not the other way around.

2. Mature the AI/LLM & Multi-Agent Layer

- Move from a single LLM call to a **structured AI pipeline**:
  - **Parser Agent**: Extracts and cleans design requirements.
  - **Engineer Agent**: Fills missing parameters, checks feasibility, refers to rules/standards.
  - **CAD Agent**: Calls the CAD library to generate models.
  - **Reviewer Agent**: Verifies design sanity and generates a short design report.
- Introduce **RAG**:
  - Ingest standards summaries and design guidelines.
  - Let agents query this knowledge instead of hard-coding everything.
- Experiment with:
  - Different LLMs and models.
  - Function calling / tools.
  - Few-shot examples for more robust extraction.

3. Build a Robust Backend API

- Design and implement a **FastAPI** service:
  - Endpoints like `/design/gear` to accept structured or natural-language requests.
  - Async processing of design jobs with job IDs and status checks.
  - File serving endpoints for generated STEP/STL.
- Add logging, error handling, and simple authentication (API keys, later OAuth).

4. Develop a User-Friendly Frontend

- Implement a web interface with:
  - Chat-style input for natural-language design requests.
  - Form-based advanced mode for explicit parameter entry.
  - 3D preview of generated gears (embedded viewer).
  - Download buttons for STEP/STL files.

- Add explanations:
  - “How I interpreted your request.”
  - “Equations and assumptions used.”

5. Extend to Additional Mechanical Components

- Identify the next high-value components:
  - Shafts, bushings, basic gear pairs/meshes.
  - Then assemblies like shaft–gear–bearing systems.
- For each new component type:
  - Add a parameter schema.
  - Implement engineering equations and constraints.
  - Implement CAD generation routines.
  - Integrate into the AI/agent system as new tools/agents.

6. Harden for SaaS: Performance, Reliability, and Security

- Optimize CAD generation performance and resource usage.
- Consider batching or queueing long-running tasks.
- Add:
  - User accounts and project storage.
  - Rate limiting and API usage tracking.
  - Basic billing/usage model (later stage).

7. Documentation, Testing, and Learning

- Maintain comprehensive documentation:
  - For engineering logic (equations, sources).
  - For AI prompts, agent roles, and assumptions.
  - For API usage and examples.
- Expand test coverage:
  - Unit tests for formulas.
  - Integration tests from text → CAD.
  - Regression tests for existing designs.
- Use the project as a continuous learning platform:
  - Try new LLM features (tool use, agents, RAG).
  - Experiment with open-source models for cost control and deeper understanding.

This document should serve as a concise blueprint: it captures your idea in detail, enumerates the core features, defines a practical tech stack, outlines a short and realistic MVP roadmap, and provides a clear sequence of steps to transform the MVP into a robust, end-to-end SaaS product.