perplexity

You are expert in Generative AI and also in architecture home floor plans. Please research approaches to achieve following workflows.

- store all floor plans with semantics description as metadata
- Retrieve floor plans matching to queries such as fetch 2 bedroom apartments with 2 washrooms
- Architectural co-pilot as an add-on to a floor plan software, where English queries are converted to command list, then it is send to APIs along with current floor plans as some text representation, then actual updation and generation gets done in the API-software, response returned.

Do you need to store floor plans in some meta structured language representation, something like json, where the whole floor plans is described in text along with dimensions such as 3 rooms, first room, Bedroom, dimensions, windows placements, etc. Any already product or research such specification? Then one should be able to RAG(retrieval augmented generation) on such data, if one asks give me all floor plans with 2 bedrooms with less than 1000 sq ft area, it should be able to retrieve those. In more complex scenario of architectural co-pilot, one should be able to interface floor plan generation and editing via English commands. Any such attempts already there in products or research? do they store meta information of floor plans, which is different than the display information which may be in svg or images.

Output crisp thoroughly researched analysis and proposal for such system.

Now I'll generate the comprehensive technical report.

Comprehensive Analysis: Generative AI Approaches for Floor Plan Management & Architectural Copilots

Executive Summary

A robust floor plan management system with architectural AI copilots requires a **three-layer architecture** combining structured semantic storage, retrieval-augmented generation (RAG), and natural language interfaces. This analysis consolidates current research, existing products, and technical standards to provide a blueprint for implementation. The system must support three core workflows: (1) **semantic storage and discovery** using graph-based metadata, (2) **floor plan retrieval via natural language queries** using RAG patterns, and (3) **architectural copilot functionality** converting English commands to API operations.

1. Structured Floor Plan Representation: Standards & Formats

1.1 Industry Foundation Classes (IFC) Standard

IFC remains the most comprehensive standard for representing architectural data with semantic richness and hierarchical structure. The standard defines a containment hierarchy that mirrors actual building organization:[1]

```
IfcProject (Top level)
  └─ IfcSite
    └─ IfcBuilding
      └─ IfcBuildingStorey (Floor level)
        └─ IfcSpace (Rooms, zones)
        └─ IfcWall, IfcDoor, IfcWindow
```

Key IFC advantages for semantic metadata:

- **IfcRelAggregates**: Hierarchical relationships (building contains floors, floors contain rooms)
- **IfcRelSpaceBoundary**: Adjacency relationships between rooms with geometric boundaries
- **IfcLocalPlacement**: 3D coordinate positioning relative to parent elements
- **Room classification**: Supports DIN 277-2:2005-02 standard for unambiguous room function identification[1]
- **Quantity data**: IFC properties store area, dimensions, and custom attributes

Practical implementation challenge: IFC is verbose and complex. Most floor plan systems convert IFC to intermediate representations for querying and modification.[1]

1.2 Graph-Based Representation

Graph structures have emerged as the **preferred intermediate format** for floor plan computation, balancing semantic expressiveness with computational efficiency.[2] [3] [4]

Graph representation pattern:

- **Nodes**: Represent spatial entities (rooms, doors, windows)
- **Edges**: Encode relationships (adjacency, containment, connectivity)
- **Node attributes**: area, dimensions, type, position coordinates
- **Edge attributes**: connection type, direction, access type

Research evidence of graph effectiveness:

- a.SCatch system uses semantic fingerprints with room-adjacency graphs for retrieval[3] [2]
- Graph Neural Networks (GNN) outperform traditional approaches for room classification using GraphSAGE and TAGCN[4]
- House-GAN++ generates floor plans by operating on input graphs constrained to room topology[5]

Example GNN-extracted node features for room classification:[4]

- Area, length, width
- Door count and placement
- Parent/child room relationships (for nested spaces)
- Adjacency list

1.3 JSON Schema for Metadata Storage

Schema.org **FloorPlan** provides a JSON-LD standard, though it lacks architectural specificity. A production system should extend this with domain-specific metadata:[6]

```
{
  "@context": "https://schema.org",
  "@type": "FloorPlan",
  "name": "2BHK Residential Unit",
  "image": "plan.svg",
  "rooms": [
    {
      "type": "bedroom",
      "area": 180,
      "length": 4.5,
      "width": 4.0,
      "windows": 2,
      "doors": 1,
      "connectedRooms": ["corridor", "ensuite"],
      "classification": "01.100" // DIN 277-2
    }
  ],
  "relationships": [
    {
      "from": "bedroom",
      "to": "bathroom",
      "type": "adjacent",
      "hasAccess": true
    }
  ]
}
```

JSON advantages: Language-agnostic, directly embeddable in vector databases, supports hierarchical structures and dependencies. Schema.org patterns standardize field definitions for interoperability.[7]

2. Semantic Retrieval: RAG for Floor Plans

2.1 RAG Architecture for Architectural Queries

Retrieval-Augmented Generation enables **natural language queries against floor plan repositories**. The workflow:[8] [9]

1. **User Query**: "Show me all 2-bedroom apartments with 2 washrooms, less than 1000 sq ft"
2. **Query Embedding**: Convert to vector via embedding model (CLIP, Sentence Transformers)
3. **Similarity Search**: Find matching floor plans in vector database
4. **Re-ranking**: Use semantic constraints (bedroom count, area) for filtering
5. **Generation**: LLM synthesizes response with retrieved plans

Vector database implementation example using Firestore:[10]

- Store floor plan metadata as documents with vector embeddings
- Create vector indexes with EUCLIDEAN distance measure
- Perform nearest-neighbor queries with pre-filtering on room counts, area ranges
- Returns relevant floor plans sorted by similarity score

Research evidence of effectiveness:

- PlanRAG method achieves 15.8% improvement over baseline RAG for decision QA through plan-then-retrieval approach[11]
- Hybrid search combining dense vectors + sparse keyword indices improves recall while maintaining precision[9]

2.2 Hybrid Search Strategy

Recommended dual-indexing approach:

Search Method	Use Case	Advantages
Vector Search (Dense)	"Modern minimalist apartments"	Captures semantic meaning, handles ambiguity
Keyword Search (Sparse)	"bedroom >= 2 AND area <= 1000"	Exact constraints, logical operators
Combined (Hybrid)	Complex queries with semantic + filters	Best precision + recall

Implementation: Pinecone or Weaviate support hybrid mode with separate dense and sparse indices, reranked by unified score.[9]

2.3 Metadata Extraction & Indexing Pipeline

Floor plans must be analyzed to extract semantic metadata for embedding:

```
Raw Floor Plan (DXF/SVG/IFC)
  ↓
[Semantic Extractor - GNN/CV]
```

```
  ↓
Structured JSON Metadata
  ↓
[Text Embedder - CLIP/E5]
  ↓
Vector Database + Metadata Store
  ↓
RAG-Ready Index
```

Multi-modal analysis needed:

- **Geometric analysis**: Room areas, dimensions, wall lengths via image/geometry processing
- **Topological analysis**: Adjacency graphs, connectivity patterns via GNN[4]
- **Semantic labeling**: Room types (bedroom, kitchen, etc.) via classification models
- **Text generation**: Natural language description of plan for embedding

3. Floor Plan Generation from Natural Language

3.1 Current State-of-the-Art Approaches

Three parallel paradigms exist, each with trade-offs:

A. Sequence-to-Sequence (Tell2Design Approach)

Dataset: Tell2Design contains 80,788 floor plans paired with natural language instructions[12]

- Input: "2-bedroom apartment with open kitchen, master ensuite, and living-dining area"
- Model: Transformer encoder-decoder (T5-based)
- Output: Structured sequence with room coordinates and dimensions

Strengths:

- Preserves explicit spatial constraints (e.g., "x coordinate = 2.5")
- Generates precise room boundaries
- Multiple outputs can be generated for same input

Limitation: Seq2Seq struggles with implicit spatial relationships and ambiguous natural language[12]

B. Graph-Constrained GANs (House-GAN++ Approach)

Architecture: Integrates graph constraints with generative adversarial networks[5]

- Input: Bubble diagram (rough layout with room adjacency)
- Constraint mechanism: Graph structure guides generation
- Output: Vector floor plans with room coordinates

Key innovation: Conv-MPN (Convolutional Message Passing Network) encodes graph topology into latent space, enabling layout refinement iteration.

Strengths:

- Produces non-rectangular room shapes
- Iterative refinement improves quality
- Handles functional graph constraints (not just adjacency)

Limitation: Requires bubble diagram input, not pure natural language

C. Diffusion Models (HouseDiffusion & FloorDiffusion)

Latest paradigm: Diffusion-based generation with Transformer control[13] [14]

- Input: Graph constraint + optional text description
- Process: Iterative denoising of room coordinates
- Output: Vector graphics floor plans

Key advantages over GANs:

- Direct vector-format output (no rasterization)
- Stable training without adversarial dynamics
- Graph attention mechanisms enforce spatial constraints
- Multiple output variants from single input

FloorDiffusion specifics: Conditional generation without paired training data requirement, makes it practical for production systems.[13]

3.2 Hybrid Text-to-Plan Workflow (Recommended)

Research demonstrates optimal performance through multi-stage pipeline:[15]

```
Natural Language Input
  ↓
[LLM: Constraint Extraction]
→ Extract rooms, adjacency, special requirements
  ↓
[GNN: Graph Construction]
→ Build bubble diagram representing constraints
  ↓
[HouseDiffusion/House-GAN++: Layout Generation]
→ Generate floor plan from constrained graph
  ↓
[Diffusion/GAN Refinement]
→ Iterative improvement based on design rules
  ↓
Final Vector Floor Plan (SVG/DXF compatible)
```

Why this works:

- LLMs excel at understanding natural language nuance and constraints
- GNNs enforce spatial consistency via graph structure
- Diffusion/GAN models handle geometric optimization
- Separates semantic understanding from spatial computation

4. Architectural Copilot: English Commands to API Operations

4.1 Existing Copilot Products & Architectures

ArchiLabs (Production Copilot in Revit)

Current implementation:[16]

- Natural language interface directly within Revit
- User input: "Create sheets for all floor plan views and add dimensions and room tags to each"
- System: Parses request → determines series of Revit API operations → executes automatically
- Technology: LLM translates intent to Revit API calls or Dynamo nodes

Critical insight: Copilots don't generate code from scratch—they **map natural language to API operations through learned patterns**, either via:

1. Prompt engineering with API documentation context
2. Fine-tuning on example command sequences
3. RAG-augmented retrieval of relevant API patterns

Maket.ai **(Constraint-Based Generation)**

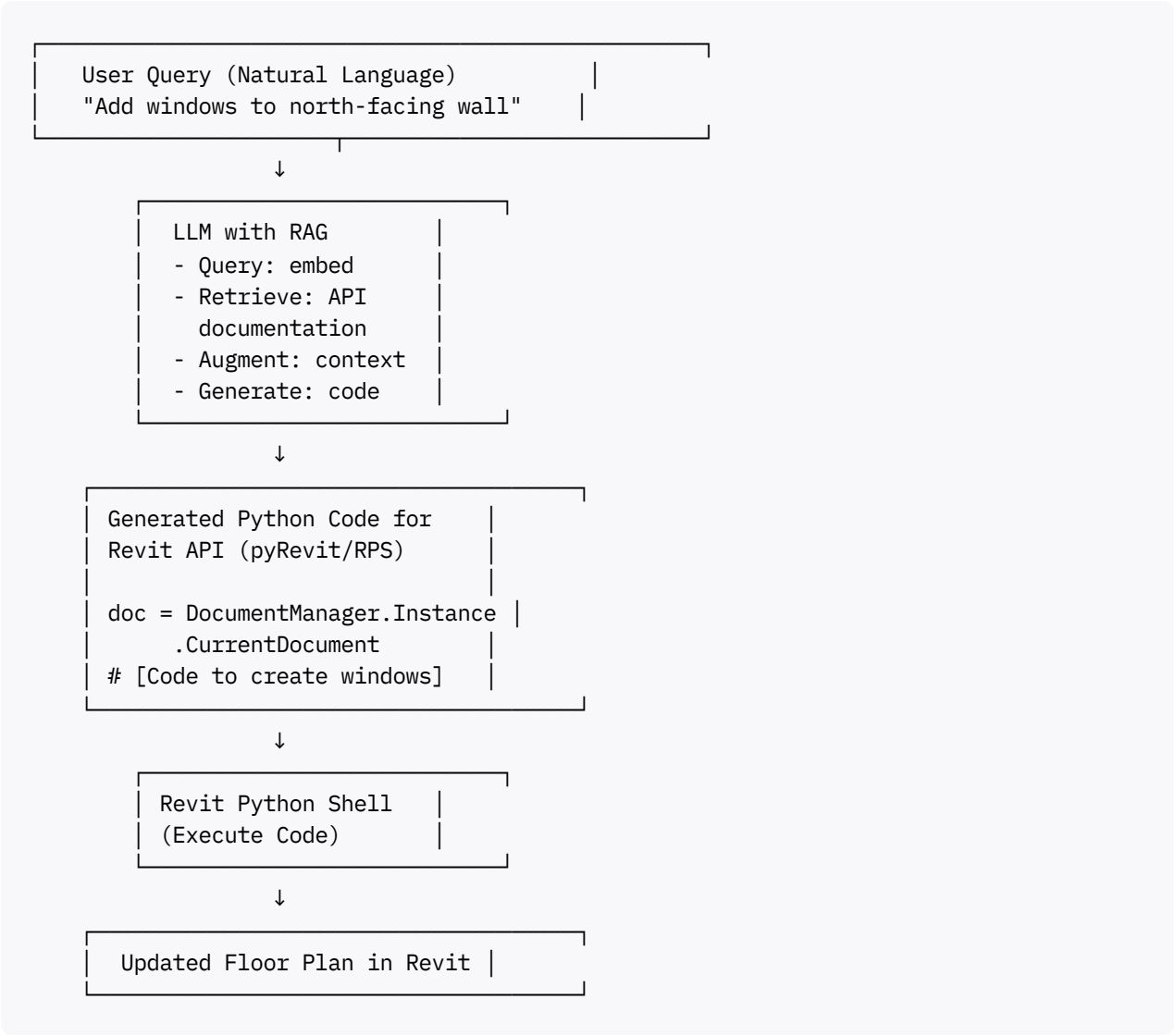
Workflow:[17]

- UI collects room dimensions, adjacency preferences
- Backend generates multiple plan variations
- Export to .DXF for CAD integration
- Natural language interface "coming soon"

Technical limitation: Current system is constraint-driven, not language-driven. Natural language layer would sit above constraint extractor.

4.2 RAG-Based Copilot Architecture (Recommended)

The most robust approach combines **RAG with Revit API knowledge**:[18]



Implementation details based on Revit automation research:[19] [18]

Automation Method	Learning Curve	Power	Best For
Revit Macros (C#)	Steep	Maximum	Deep integrations, performance-critical
Dynamo (Visual)	Gentle	Medium	Non-programmers, reusable workflows
Python (pyRevit)	Moderate	High	Developers, team deployments
AI Copilot (LLM)	None	Medium-High	End-users, natural exploration

RAG context construction for Revit copilots:

1. Index Revit API documentation (classes, methods, properties)
2. Include example code snippets from pyRevit library
3. Store common patterns (e.g., "iterate rooms", "place door family")
4. Retrieval: When user query comes in, fetch relevant API snippets + patterns

5. LLM augmented with context generates Python code
6. Code executes in RevitPythonShell

4.3 Plan-to-Modification Workflow

For architectural copilot to modify existing floor plans:

```
English Command
  ↓
[Intent Classification]
→ Is this: creation? modification? deletion? constraint? analysis?
  ↓
[Parameter Extraction]
→ Parse rooms, areas, adjacencies, dimensions
  ↓
[Floor Plan Context]
→ Load current plan metadata (graph representation)
  ↓
[Modification Engine]
→ Modify graph structure / regenerate affected areas
  ↓
[Validation]
→ Check code compliance, spatial feasibility
  ↓
[Visualization Update]
→ Regenerate SVG/DXF, update Revit model
```

Key technical challenge: Distinguishing between **valid constraints** (must enforce) vs. **design preferences** (can relax). This requires domain-specific reasoning, currently best achieved through prompt engineering with architectural guidelines.

5. Data Format Conversions & Interoperability

5.1 Format Landscape

Floor plans exist in multiple representations, each optimized for different purposes:

Format	Primary Use	Metadata Support	Semantic Richness	Conversion Difficulty
IFC	BIM software (Revit, ArchiCAD)	Excellent	Excellent	Easy from→ Hard to←
DXF	CAD software (AutoCAD)	Minimal	Minimal	Moderate
SVG	Web visualization	Moderate (custom)	Low	Moderate
JSON	APIs, databases	Excellent (custom)	Depends on schema	Easy

Format	Primary Use	Metadata Support	Semantic Richness	Conversion Difficulty
Raster (PNG/JPG)	Human viewing, input	None	None	Hard (OCR)

5.2 Conversion Pipeline (Recommended)

```
Source Format (DXF, IFC, Image)
  ↓
[Extraction Layer]
  ├─ Geometric: walls, doors, windows, furniture
  ├─ Topological: room adjacency, connectivity
  └─ Semantic: room labels, dimensions, materials
  ↓
[JSON Standardization]
→ IFC-compatible JSON wrapper with schema.org extensions
→ Embed room types (DIN 277-2 codes)
→ Structure as graph representation
  ↓
[Storage]
├─ Graph DB (Neo4j): Topology + relationships
├─ Document DB (MongoDB): Full JSON + metadata
└─ Vector DB (Pinecone): Embeddings for retrieval
  ↓
[Query/Generation]
  ├─ RAG retrieval via natural language
  ├─ Modification via copilot commands
  └─ Export to target format (SVG, DXF, PDF)
```

Practical tools for conversions:

- **DXF → GeoJSON**: QGIS with coordinate system projection[20]
- **JSON → DXF**: Aspose library or custom conversion scripts[21]
- **DXF → SVG**: ConvertAPI with layer/space selection[22]
- **Automated extraction**: Computer vision + graph neural networks for image → JSON

6. Recommended System Architecture

6.1 Three-Layer Architecture Diagram

Proposed Three-Layer Architecture for Floor Plan Management System

6.2 Storage Layer Design

Multi-database approach balances query flexibility with performance:

```
┌─────────────────────────────────────────────────────┐
│         STORAGE LAYER                               │
├─────────────────────────────────────────────────────┤
│                                                     │
│  Graph Database (Neo4j)                             │
│  ├─ Room nodes with properties                      │
│  ├─ Adjacency/containment edges                     │
│  └─ Fast topology queries                           │
│                                                     │
│  Vector Database (Pinecone/Weaviate)                │
│  ├─ Floor plan embeddings                           │
│  ├─ Semantic similarity search                      │
│  └─ Hybrid keyword indexing                         │
│                                                     │
│  Document Store (MongoDB/Firestore)                 │
│  ├─ Full JSON metadata documents                    │
│  ├─ Historical versions/audit trail                 │
│  └─ Custom indexing                                 │
│                                                     │
│  File Storage (S3/GCS)                              │
│  ├─ SVG/DXF exports                                 │
│  ├─ Raster visualizations                           │
│  └─ Source documents                                │
│                                                     │
└─────────────────────────────────────────────────────┘
```

Query routing logic:

- Structural queries ("rooms adjacent to kitchen") → Neo4j
- Semantic queries ("modern minimalist plans") → Vector DB + RAG
- Metadata queries ("all plans with area < 1000") → Document DB
- Complex queries → Combine across databases

6.3 Generation & Retrieval Layer

Modular microservices approach:

```
Floor Plan Generation Service
├─ Text-to-constraints (LLM)
├─ Graph construction (GNN)
├─ Layout generation (HouseDiffusion/House-GAN++)
└─ Refinement engine (Validation + iteration)

RAG Retrieval Service
├─ Query embedding (CLIP/E5)
├─ Vector similarity search
├─ Constraint filtering
└─ Re-ranking + augmentation
```

```
Modification Service
├── Plan parsing (DXF/IFC → JSON)
├── Constraint application
├── Topology updates
└── Re-generation of affected areas
```

6.4 Interaction Layer (Copilot)

Natural language processing pipeline:

```
User Input (English)
  ↓
[Intent Recognition]
→ Create / Modify / Retrieve / Analyze
  ↓
[Entity Extraction]
→ Room types, quantities, dimensions, constraints
  ↓
[Plan Context Loading]
→ Retrieve current floor plan if modifying
  ↓
[RAG-Augmented Generation]
→ For Revit: Generate Python code via Revit API RAG context
→ For generation: Generate floor plan via generation service
  ↓
[Execution & Feedback]
→ Run code / Generate plan
→ Display result + confidence metrics
→ Log action for audit trail
```

7. Semantic Metadata Standard (Proposed JSON Schema)

7.1 Core Schema Structure

```
{
  "$schema": "https://example.com/floor-plan-schema/v1.0",
  "$id": "floor-plan-residential-unit-001",
  "metadata": {
    "version": "1.0",
    "created": "2024-01-15",
    "modified": "2024-01-15",
    "createdBy": "architect_system",
    "source": "Revit export"
  },
  "building": {
    "name": "Residential Tower Block A",
    "address": "123 Main St, City",
    "totalArea": 45000,
    "areaUnit": "sqft"
  },
```

```
  "floorPlan": {
    "name": "Typical Unit - Floor 5",
    "level": 5,
    "totalArea": 950,
    "rooms": [
      {
        "id": "room_001",
        "type": "bedroom",
        "dinCode": "01.100.0010",
        "area": 180,
        "dimensions": {
          "length": 4.5,
          "width": 4.0,
          "unit": "m"
        },
        "features": {
          "windows": 2,
          "doors": 1,
          "balcony": true
        },
        "geometry": {
          "vertices": [[0, 0], [4.5, 0], [4.5, 4.0], [0, 4.0]],
          "coordinates": "cartesian"
        }
      }
    ],
    "adjacencies": [
      {
        "room1": "room_001",
        "room2": "room_bathroom_001",
        "type": "access",
        "hasWall": false,
        "description": "direct access to ensuite"
      }
    ],
    "zones": [
      {
        "name": "sleeping_zone",
        "rooms": ["room_001", "room_bedroom_002"],
        "separationLevel": "private"
      }
    ]
  },
  "constraints": {
    "codRequirements": ["natural_light_minimum", "ventilation_minimum"],
    "designNotes": "Maximize cross-ventilation, minimize noise"
  }
}
```

Key semantic fields:

- **DIN code**: Standard room classification for unambiguous identification
- **Adjacency relations**: With descriptive attributes (access type, wall presence)
- **Zone groupings**: Functional areas (sleeping, service, living)

- **Constraints**: Compliance requirements and design goals
- **Geometry**: Both symbolic (vertices) and semantic (type, area)

7.2 Schema Extensibility

Support inheritance and custom properties:

- **Extension fields**: `x-custom:*` for domain-specific metadata
- **Versioning**: Multiple schema versions for backward compatibility
- **Localization**: Room type names in multiple languages
- **Material/finish data**: Linkable via separate material database

8. Comparison: Representation Approaches

Comparison of Floor Plan Representation Approaches Across Key Dimensions

Key insight: IFC wins on compliance and BIM integration, but graph-based JSON is superior for retrieval and generation workflows. **Optimal approach**: Store authoritative data in IFC for compliance, but maintain JSON graph representation for operational queries and AI operations.

9. Implementation Roadmap

Phase 1: Foundation (Months 1-3)

- [ ] Define JSON schema for floor plan metadata (extend schema.org)
- [ ] Establish graph database (Neo4j) with room/adjacency schema
- [ ] Build metadata extraction pipeline (DXF/IFC → JSON)
- [ ] Implement vector embedding + Pinecone indexing
- [ ] Basic RAG retrieval for floor plans ("find plans with 2+ bedrooms")

Phase 2: Generation (Months 4-6)

- [ ] Integrate Tell2Design-based Seq2Seq model (or HouseDiffusion)
- [ ] Build natural language → graph/constraint extractor
- [ ] Implement validation engine (code compliance, spatial feasibility)
- [ ] Create visualization/export pipeline (SVG, DXF, PDF)

Phase 3: Copilot (Months 7-9)

- [ ] RAG integration with Revit API documentation
- [ ] Python code generation via LLM + context injection
- [ ] pyRevit execution framework with error handling
- [ ] Plan modification workflow (preserve unchanged areas)
- [ ] Audit logging and version control

Phase 4: Optimization (Months 10-12)

- [ ] Performance tuning (vector DB indexing, query optimization)
- [ ] Multi-user collaborative editing
- [ ] Mobile/web UI for non-technical users
- [ ] Integration with cloud BIM platforms (Speckle, Archbase)

10. Key Technologies & Products Evaluated

Technology	Maturity	Relevance	Recommendation
Tell2Design Dataset	Production	Benchmark + training	Use for baseline models
House-GAN++ / HouseDiffusion	Research → Production	Plan generation	Implement HouseDiffusion for stability
Revit Python API	Mature	Copilot execution	Use pyRevit for team deployment
Pinecone / Weaviate	Production	Vector search	Pinecone for managed, Weaviate for on-prem
Neo4j	Mature	Graph queries	Proven for spatial relationships
CLIP / E5 Embeddings	Mature	Query embedding	E5-large for domain robustness
ArchiLabs Copilot	Beta (Commercial)	Reference implementation	Study architecture, consider partnership
Maket.ai **/ Laiout**	Production	Constraint-based generation	Reverse-engineer UI/UX patterns

11. Critical Gaps & Research Opportunities

Outstanding challenges identified:

1. **Semantic→Geometric mapping ambiguity**: "Open concept living-dining area" admits multiple valid layouts. Current models struggle with design intent beyond spatial constraints.
2. **Code compliance automation**: Embedding building codes into generation models remains nascent. Current practice requires post-generation validation.

3. **Iterative collaborative design**: Systems excel at single-shot generation, not multi-turn refinement with architect feedback loops.
4. **Cross-cultural design patterns**: Models trained on Western layouts may fail for other architectural traditions (feng shui adjacencies, courtyard-centric designs, etc.).
5. **Bidirectional IFC↔JSON conversion**: No mature open-source tooling exists. Most systems implement one-way (IFC→JSON).

12. Conclusion & Strategic Recommendation

Build a modular, API-first platform with these architectural decisions:

1. **Storage**: Graph (Neo4j) + Vector (Pinecone) + Documents (Firestore)—**not monolithic** IFC storage
2. **Generation**: Hybrid LLM + GNN + diffusion pipeline—**not single model**
3. **Metadata**: JSON wrapper around IFC concepts—**enable both standards compliance and AI operations**
4. **Copilot**: RAG + Revit API context injection—**practical over perfect**
5. **Format flexibility**: DXF/IFC input, SVG/JSON output—**multiple entry/exit points**

This design prioritizes **semantic searchability and generation capability over strict standards compliance**, while maintaining eventual IFC compatibility. It mirrors current production patterns (Maket, Laiout, ArchiLabs) and aligns with academic research evidence (House-GAN++, HouseDiffusion, Tell2Design).

The system becomes valuable when it handles the **"second meter" of design**—not just initial sketch generation, but continuous refinement through natural language interaction, constraint satisfaction, and code compliance verification.

⁂

1. https://d-nb.info/1301846449/34
2. https://mediatum.ub.tum.de/doc/997905/997905.pdf
3. https://www.dfki.de/fileadmin/user_upload/import/9805_54-Qamer-Floor-Plan-Searching-ICPRAM17.pdf
4. https://arxiv.org/pdf/2108.05947.pdf
5. https://openaccess.thecvf.com/content/CVPR2021/papers/Nauata_House-GAN_Generative_Adversarial_Layout_Refinement_Network_towards_Intelligent_Computational_Agent_CVPR_2021_paper.pdf
6. https://schemantra.com/schema_list/FloorPlan
7. https://utrechtuniversity.github.io/yoda/design/metadata/metadata-form-json.html
8. https://aws.amazon.com/what-is/retrieval-augmented-generation/
9. https://www.pinecone.io/learn/retrieval-augmented-generation/
10. https://firebase.google.com/docs/firestore/vector-search
11. https://aclanthology.org/2024.naacl-long.364/

12. https://aclanthology.org/2023.acl-long.820.pdf
13. https://www.sciencedirect.com/science/article/abs/pii/S2352710224018886
14. https://openaccess.thecvf.com/content/CVPR2023/papers/Shabani_HouseDiffusion_Vector_Floorplan_Generation_via_a_Diffusion_Model_With_Discrete_CVPR_2023_paper.pdf
15. https://mgv.sggw.edu.pl/article/download/10294/9190/19916
16. https://archilabs.ai/posts/google-gemini-3-for-architecture
17. https://www.maket.ai
18. https://www.linkedin.com/posts/ismailseleit_ai-rag-revit-activity-7315823533559828480-zPTE
19. https://archilabs.ai/posts/revit-automation-macros-vs-dynamo-vs-python-vs-ai
20. https://www.reddit.com/r/gis/comments/1bl2v86/advice_on_converting_cad_drawing_to_geojson/
21. https://products.aspose.com/total/net/conversion/json-to-dxf/
22. https://www.convertapi.com/dxf-to-svg
23. https://developer.hashicorp.com/terraform/internals/json-format
24. https://docs.spring.io/spring-ai/reference/api/retrieval-augmented-generation.html
25. https://docs.cloud.google.com/storage/docs/metadata
26. https://www.scitepress.org/Papers/2017/61128/index.html
27. https://www.labkey.org/Documentation/wiki-page.view?name=metadataJson
28. https://caadria2021.org/wp-content/uploads/2021/03/caadria2021_151.pdf
29. https://stackoverflow.com/questions/57216366/how-to-represent-meta-data-in-json
30. https://www.sciencedirect.com/science/article/pii/S0926580524003856
31. https://www.k2view.com/what-is-retrieval-augmented-generation
32. https://arxiv.org/html/2509.00543v1
33. https://architizer.com/blog/practice/tools/top-ai-tools-for-architects-and-designers/
34. https://www.microsoft.com/en-us/microsoft-copilot/copilot-101/ai-for-architecture
35. https://camplan.ai
36. https://www.archivinci.com/architecture-ai-tools/archigpt
37. https://arxiv.org/abs/2311.15941
38. https://blog.alicetechnologies.com/news/best-ai-tools-for-architects-in-2025-a-comprehensive-guide
39. https://www.youtube.com/watch?v=m7EuZ7GhinE
40. https://arxiv.org/html/2410.11908v1
41. https://mnml.ai
42. https://architizer.com/blog/practice/tools/top-ai-tools-for-generating-architectural-plans/
43. https://ai.google.dev/competition/projects/dream-design
44. https://www.aecbytes.com/review/2024/SketchProAI.html
45. https://www.linkedin.com/pulse/ifc-schema-explained-sidath-jayasingha-cstrf
46. https://papers.cumincad.org/data/works/att/caadria2024_497.pdf
47. https://docs.open-metadata.org/latest/developers/contribute/developing-a-new-connector/define-json-schema
48. https://revizto.com/en/what-is-ifc-format/

49. https://www.semanticscholar.org/paper/Room-Classification-on-Floor-Plan-Graphs-using-Paudel-Dhakal/b72721c39aff623a57ed4d8dcf3cf8790705b24f
50. https://brickschema.org/papers/shepherding2020fierro.pdf
51. https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcSpace.htm
52. https://rvsn.csail.mit.edu/content/Pubs/master_whiting_2006june_bmgwriteup.pdf
53. https://www.sciencedirect.com/science/article/abs/pii/S0378778824011551
54. https://itc.scix.net/pdfs/w78-2010-43.pdf
55. https://discourse.mcneel.com/t/extracting-structural-and-skeleton-graphs-from-floor-plans/185632
56. https://annex81.iea-ebc.org/Data/publications/IEA Annex 81 Survey of Metadata Schemas.pdf
57. https://www.youtube.com/watch?v=zYJ1SA_Rpq0
58. https://www.qwak.com/post/utilizing-llms-with-embedding-stores
59. https://www.semanticscholar.org/paper/Tell2Design:-A-Dataset-for-Language-Guided-Floor-Leng-Zhou/04a0f3f39c5a6aeddcd2841457d0b500adfcb534
60. https://nexla.com/ai-infrastructure/vector-embedding/
61. https://caadria2024.org/wp-content/uploads/2024/04/166-CHATDESIGN.pdf
62. https://www.youtube.com/watch?v=X9z4E_AsFNw
63. https://arxiv.org/html/2509.03737v1
64. https://github.com/ChatHouseDiffusion/chathousediffusion
65. https://www.echeverrimontes.com/en/blog/python-tips-for-automating-processes-in-revit
66. https://www.dailydoseofds.com/a-beginner-friendly-and-comprehensive-deep-dive-on-vector-databases/
67. https://www.ijnrd.org/papers/IJNRD2303434.pdf
68. https://stackoverflow.com/questions/41715557/convert-dxf-file-to-json-geometry-4
69. https://www.youtube.com/watch?v=X5654jxAatw
70. https://www.sciencedirect.com/science/article/pii/S2352710223005570
71. https://2050-materials.com/blog/laiout-automated-floor-plans-powered-by-2050-materials/
72. https://dl.acm.org/doi/abs/10.1145/3386569.3392391
73. https://www.laiout.co
74. https://conf.dap.tuwien.ac.at/preprints/ecaade2025/ecaade2025_272.pdf
75. https://jdrfelectromag.github.io/dxf-converter-ui-prototype/
76. https://gtf.fyi/papers/fierro_masters_thesis.pdf
77. https://papers.cumincad.org/data/works/att/sigradi2021_200.pdf
78. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-106.pdf
79. https://ennauata.github.io/housegan/housegan_suppl.pdf
80. https://synaptica.com/relationships-in-ontology-design/
81. https://ar5iv.labs.arxiv.org/html/2108.05947
82. https://www.sciencedirect.com/science/article/pii/S1474034625006548
83. https://arxiv.org/html/2405.17236v1