

Site-Specific Architectural Design of Multi-Story Residential Buildings with Hierarchical Expert Iteration

Markus Hauser ^{1*} and Heike Hauser¹

^{1*}Minonexus GmbH, Isartalstraße 32, Munich, 80469, Germany.

*Corresponding author(s). E-mail(s): markus@minonexus.com;
Contributing authors: heike@minonexus.com;

Abstract

Purpose: Residential floor-plan design is a combinatorial problem without a closed-form solution. Deep generative models mostly produce images or coarse room boxes of single-story apartments without reference to the plot, and general-purpose large language models lack robust geometric reasoning. We investigate how preliminary designs for detached houses of one or more stories can be generated as editable building models on real plots.

Methods: We present MinoSketch, a deployed system for plots in Germany. A large language model elicits requirements in natural language. A Bayesian network learned from 168 captured architectural designs drives a masked sequential sampler that yields consistent room programs. Open geodata on parcels, buildings, terrain, and development plans constrain and inform the design. A hierarchical design engine, structured into options, is trained by Expert Iteration in synthetic self-play against a cost function developed with architects. Metropolis sampling, mixed-integer optimization, and deep reinforcement learning were evaluated and abandoned.

Results: On 60 fixed design tasks, the engine produces valid designs in 97–98% of cases. The share of self-play episodes reaching a valid design rose from about 63% to 95%. Masked sampling eliminated over-filled stories in the room programs.

Conclusions: A search-based, hierarchical, and trainable approach can generate site-specific designs without public training data. The deployed system exports building models in the Industry Foundation Classes format (IFC 4x3) for professional workflows.

Keywords: generative design, floor-plan generation, Monte Carlo tree search, expert iteration, open geodata, BIM

1 Introduction

Somebody who wants to build a house usually has requirements but no picture of how they would materialize on their own plot. The first meeting with an architect or a construction company therefore typically starts from a list of wishes. A preliminary design that already reflects those wishes *and* the plot—its shape, terrain, neighboring buildings, and the applicable building regulations—would help clients clarify their wishes and see what can be realized on their specific plot, and it would make this conversation considerably more concrete.

Producing such a design automatically is hard. Space layout planning has been studied for more than five decades, and no precise general solution exists (Lobos & Donath, 2010). A design must satisfy hard constraints (all rooms reachable, no overlaps, a staircase that fits and is aligned across stories, compliance with the buildable area) and simultaneously trade off many soft criteria (target areas and proportions, adjacencies, orientation, daylight, privacy gradients, circulation efficiency). Multi-story houses add the problem of vertical circulation: the position of the staircase couples the layouts of all floors and is among the most design-determining early decisions.

Large language models (LLMs) are increasingly used by laypeople to produce images of building designs, but text-only models do not possess robust spatial or geometric reasoning. Image-generating models, in turn, yield pictures rather than dimensioned, editable building elements. For a design that can be measured, modified, and handed over to an architect’s computer-aided design (CAD) workflow, the result must be a geometric building model.

In this paper we describe *MinoSketch*, a system that generates preliminary designs for detached single-family houses on concrete plots in Germany and that has been available as a browser-based service since 2026. The system works in two stages: it first derives a room program from the user’s requirements and the site (Stage I) and then generates the geometric design for this program on the plot (Stage II). Our contributions are:

1. a formulation of *design in context*, in which open geodata about the plot and its surroundings enter both the requirement model and the design search (Section 4);
2. a room-program model based on a Bayesian network learned from a small corpus of captured designs, combined with a masked sequential sampler that guarantees consistent programs (Stage I, Section 6);
3. a report on alternative approaches to layout generation that we evaluated and abandoned, which we hope is useful to others working on the problem (Stage II, Section 7);
4. a hierarchical design engine that mirrors the professional design process—circulation, zoning, room definition, refinement—as a hierarchy of options trained by Expert Iteration in synthetic self-play (Stage II, Section 8); and
5. first results from a deployed system that exports models in the Industry Foundation Classes format (IFC 4x3; ISO, 2024) (Section 9).

MinoSketch is developed and operated commercially. To protect proprietary know-how, we describe the method at a conceptual level and do not disclose all implementation details. We nevertheless aim to contribute to the research community by

sharing the architecture of the approach and our practical experience, including the approaches that did not work (Section 7).

2 Related Work

Space layout planning

Automated space allocation has been approached with constraint-based systems, optimization, shape grammars, evolutionary algorithms, physically based models and, more recently, machine learning; Lobos and Donath (2010) give a survey. Early work by Flemming established *rectangular dissections* and their wall representation as a foundation for systematic layout generation (Flemming, 1978). Dissection-based derivations remain a standard way of refining a building outline step by step, but they produce strictly rectangular rooms. Practical designs therefore also require primitives that move wall segments, as in the residential layout generator of Merrell, Schkufza, and Koltun (2010), which couples a Bayesian network for room programs with stochastic optimization of the geometry. Our work shares this two-stage structure but replaces the stochastic optimization with a learned, hierarchical search.

Generative deep learning

A large body of work generates floor plans with deep generative models, for example from bubble diagrams with relational generative adversarial networks (GANs; Nauata, Chang, Cheng, Mori, & Furukawa, 2020) or from layout graphs (Hu et al., 2020); see C. Li, Zhang, Du, Zhang, and Xie (2024) for an overview of generative AI in architectural design. These methods predominantly produce raster images or coarse box layouts that lack the precision required for further processing in CAD, and they are trained on large datasets that are not available for our setting (see below).

Parametric and rule-based design tools

In professional practice, generative design is mostly realized on parametric modeling platforms. Grasshopper (Robert McNeel & Associates, n.d.-a), the visual programming environment of Rhino (Robert McNeel & Associates, n.d.-b), is a white-box environment in which designers build their own algorithms. Design rules and layout templates are encoded in scripts (Zhang, 2025), and plug-ins couple parametric models with generic optimization engines such as evolutionary solvers (Z. Li, Li, Hinchcliffe, Maitless, & Birbilis, 2024). Academic floor-plan generators have been built in the same way, e.g., on Grasshopper (Nisztuk & Myszkowski, 2019) or on Dynamo for Revit (Das, Day, Hauck, Haymaker, & Davis, 2016). Commercial assistants such as Finch (Finch3D, n.d.) follow a related approach. As we know of no peer-reviewed description, we rely on their public documentation: outlines are subdivided and circulation is laid out from libraries of predefined elements and templates, adapted by rule-based local optimization, and linked parametrically to Revit or Rhino/Grasshopper. In these tools, design knowledge resides in handwritten scripts, templates, and rules; adaptivity results from parametrization rather than from learning, and their use requires design expertise on the part of the user.

Site context in early design

A growing number of platforms bring site data—location, terrain, access, neighboring buildings, regulations—into early design phases, for example Autodesk Forma ([Autodesk, n.d.](#)) or platforms supporting real-estate decisions such as Syte ([syte GmbH, n.d.](#)). Where they support design, they do so at the level of massing studies: building volumes are sketched and evaluated in 3D, while the internal organization of the building—rooms, circulation, stories—is not generated. MinoSketch draws on comparable data (Section 4) but uses it to condition both the room program and the floor-plan design.

LLMs and CAD

Since text-only LLMs lack spatial reasoning, current research couples them with external tools, for instance by translating prompts into scripts executed in Rhino/Grasshopper or Autodesk Revit; CAD vendors pursue similar directions. These tools assist professional users with scripting. Our goal is different: an assistant for clients without design training. In MinoSketch, the LLM is therefore restricted to dialogue and requirement elicitation, while geometry is produced by a dedicated design engine.

Datasets

Public floor-plan datasets such as RPLAN (predominantly single-level apartments in Asia; [Wu et al., 2019](#)), CubiCasa5K (mainly raster images; [Kalervo, Ylioinas, Häikiö, Karhu, & Kannala, 2019](#)) and the Modified Swiss Dwellings dataset (apartment complexes without site and geodata; [van Engelenburg et al., 2024](#)) do not cover multi-story detached houses in their geographic context. Many learned approaches consequently operate on a single level and leave out vertical circulation and the arrangement of stories—precisely the aspects that dominate the design of a single-family house. Architects rarely share design data so far—although a recent survey of German architects reports a high willingness to share data to improve AI models ([Bundesarchitektenkammer, 2025](#))—and existing material is often in pixel form without geolocation. This lack of training data is a central motivation for the self-play approach described in Section 8.5.

Search and learning

Monte Carlo tree search (MCTS; [Kocsis & Szepesvári, 2006](#)) combined with learned policy and value networks achieved super-human performance in Go through self-play ([Silver et al., 2017](#)). Expert Iteration (ExIt; [Anthony, Tian, & Barber, 2017](#)) formalizes the underlying idea: a search-based expert generates improved targets from which a neural apprentice learns, and the apprentice in turn guides the expert. The options framework of [Sutton, Precup, and Singh \(1999\)](#) provides temporally extended actions (see also [Sutton & Barto, 2018](#), Sec. 17.2), which we use to model the hierarchical nature of design.

Positioning

To our knowledge, no commercial or academic system (i) is usable by clients without design training, (ii) addresses residential buildings with a strong need for individualization rather than standardized commercial buildings, (iii) acquires its design competence through training—by self-play against a cost function developed with architects and from a corpus of captured designs—rather than from predefined templates and scripts, and (iv) generates multi-story floor plans in their site context. Research prototypes typically evaluate individual methods on single-story dwellings or apartments, often with restrictions on the number and shape of rooms and without connection to the site (e.g., [Wu et al., 2019](#); [Zawidzki & Szklarski, 2020](#)). Table 1 summarizes the comparison.

Table 1 Positioning of MinoSketch relative to classes of existing approaches (as far as publicly documented)

Approach	Users	Output	Multi-story	Site context	Design knowledge
Parametric scripting, optimization	professionals	model	per script	per script	scripts, rules
Template-based assistants	professionals	model	yes	limited	templates, rules
Site platforms	professionals, investors	massing	—	yes	—
Learned generative models	research	images, boxes	mostly no	no	datasets
MinoSketch	clients	IFC model	yes	yes	cost function, corpus, self-play

Examples: parametric scripting/optimization ([Nisztuk & Myszkowski, 2019](#); [Robert McNeel & Associates, n.d.-a](#)); template-based assistants ([Finch3D, n.d.](#)); site platforms ([Autodesk, n.d.](#); [syte GmbH, n.d.](#)); learned generative models ([Hu et al., 2020](#); [Nauata et al., 2020](#))

3 Problem Statement

We consider the generation of a building model X for a detached house of one or more stories. Two inputs determine it:

- a *site context* $\mathcal{S}$: plot geometry, buildable area, regulatory parameters (site coverage ratio, German *Grundflächenzahl*, GRZ: the share of the plot area that may be covered by buildings; floor area ratio, *Geschossflächenzahl*, GFZ: the permitted total floor area of all full stories relative to the plot area; maximum number of stories), terrain, north direction, street access, and neighboring buildings; and
- a *room program* $\mathcal{P}$: building envelope parameters (number of stories, footprint class, roof type), the set of rooms with type, story, target area, and a flag marking each room as required or optional, and a target topology of required or preferred connections between rooms (door, open connection, or passage).

The site context is available from open geodata (Section 4); the room program is not. We therefore decompose the task into two stages, which the remainder of the paper follows.

Stage I: room program. The user states requirements in natural language rather than as a program. Let e denote the evidence extracted from the dialogue and from $\mathcal{S}$. Stage I draws a program that is plausible given e and satisfies hard feasibility rules h_k , such as the area budget of each story:

$$\mathcal{P} \sim p(\mathcal{P} \mid e, \mathcal{S}) \quad \text{subject to} \quad h_k(\mathcal{P}; \mathcal{S}) = 1 \quad \forall k. \quad (1)$$

Stage II: geometric design. Given $\mathcal{P}$ and $\mathcal{S}$, Stage II generates the building model. A model X consists of stories, rooms, wall segments, openings, a staircase, and a roof. We distinguish hard validity constraints $g_j(X; \mathcal{P}, \mathcal{S}) \in \{0, 1\}$ and a soft cost $\Phi(X; \mathcal{P}, \mathcal{S}) \geq 0$. The task is

$$X^* \in \arg \min_{X \in \mathcal{V}(\mathcal{P}, \mathcal{S})} \Phi(X; \mathcal{P}, \mathcal{S}), \quad \mathcal{V}(\mathcal{P}, \mathcal{S}) = \{X : g_j(X; \mathcal{P}, \mathcal{S}) = 1 \quad \forall j\}. \quad (2)$$

The design space is mixed: it contains continuous quantities (room dimensions, wall positions) and discrete decisions (assignment of a room type to a functional zone, choice of circulation archetype, staircase type).

Stage I determines *what* is to be built and is learned from captured designs (Section 6). Stage II determines *how* it is arranged and is solved by search against a cost function (Sections 7 and 8).

4 Design in Context: Open Geodata

A floor plan without a plot is arbitrary. Before any geometry is generated, MinoSketch assembles the site context from official open data provided by the German federal states:

- **cadastral parcels** (ALKIS, *Amtliches Liegenschaftskatasterinformationssystem*, the official German real-estate cadaster information system): plot boundary, area, and street access;
- **3D building models** (LoD2, level of detail 2): heights, roof shapes, and distances of neighboring buildings;
- **digital terrain models** (DGM1, *Digitales Geländemodell*, 1 m grid): slope and ground-floor elevation;
- **development plans** (*Bebauungspläne*, legally binding municipal land-use plans; partly available in XPlanung, the German exchange standard for spatial plans; [XLeitstelle](#), n.d.): permitted use, GRZ, GFZ, number of stories, roof form; for development plans alone, data from more than 80 geodata services are integrated;
- **census grid data** (Zensus 2022) as weak prior information on household composition.

Fig. 1 shows how parcel, development plan, and terrain are presented to the user.

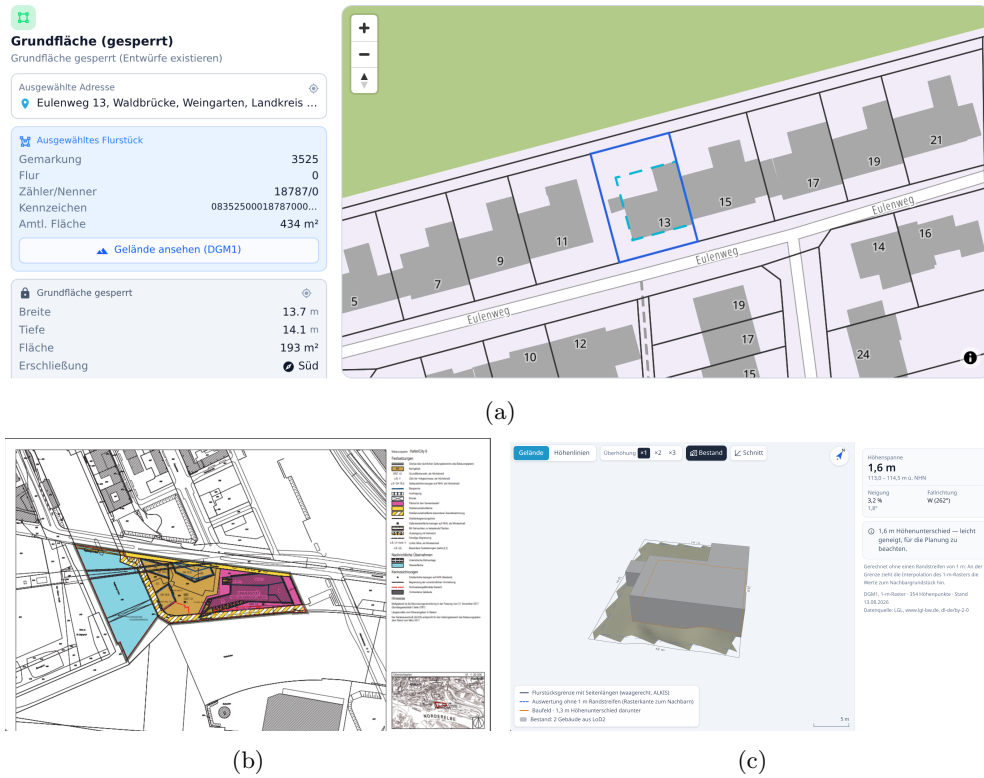


Fig. 1 Open geodata of the site. The user interface is in German; here and in the following figures, key terms are translated in the caption. (a) Cadastral parcel (ALKIS) with the footprint marked by the user and the derived plot data; *Ausgewähltes Flurstück*: selected parcel; *Gemarkung*: cadastral district; *Amtl. Fläche*: official area; *Grundfläche*: footprint; *Breite/Tiefe*: width/depth; *Erschließung*: access side. (b) Development plan of another site (Hamburg), shown with its legend; the plan fixes, among others, GRZ, number of stories, and building heights. (c) Terrain of the plot in (a) from DGM1 with the existing buildings from LoD2; *Gelände*: terrain; *Höhenlinien*: contour lines; *Überhöhung*: vertical exaggeration; *Bestand*: existing buildings; *Schnitt*: section; *Höhenspanne*: elevation range; *Neigung*: slope; *Fallrichtung*: aspect; *Baufeld*: buildable area

These data enter the design in three ways. First, they *constrain*: the buildable area, footprint, and story limits bound the envelope and the total floor area. Second, they *inform*: where no development plan exists, the German Building Code (§ 34 BauGB) requires a new building to fit into its surroundings. We therefore derive statistics of the neighboring buildings (Fig. 2); a dominant roof form or story count among distance-weighted neighbors becomes evidence for the room-program model. Third, they *situate*: north direction and street side define the orientation targets of rooms and the entrance, and the final model is georeferenced and placed on the terrain.



Fig. 2 Statistics of the surrounding buildings derived from open LoD2 data (45 buildings within 70m: heights, roof forms, footprints). Where no development plan exists, such statistics inform the design. *Umgebungsbebauung*: surrounding buildings; *Gebäudehöhen*: building heights; *Dachformen*: roof forms (*Flachdach*: flat, *Mischform*: mixed, *Satteldach*: gable, *Pultdach*: mono-pitch); *Grundflächen*: footprints

5 System Overview

Fig. 3 shows the pipeline. The user selects a plot on a map and describes the desired house in a dialogue with an LLM-based assistant (Fig. 4). The assistant extracts structured requirements—for example the number of children’s rooms, a home office on the ground floor, or a minimum kitchen size—and asks back only in case of conflicts. Requirements and site evidence condition the room-program model (Section 6), whose output, a consistent room program, is shown to the user as a bubble diagram. Together with the site context, the program forms a *design request* that is processed asynchronously by the design engine (Section 8). The result comprises floor plans per story, a 3D model placed between the real neighboring buildings, and an IFC 4x3 export for CAD systems. The system is also accessible from general-purpose AI assistants via the Model Context Protocol (MCP).

The remainder follows the two stages. Section 6 addresses Stage I. Sections 7 and 8 address Stage II: Section 7 reports the approaches to layout generation we evaluated and motivates the search-based engine described in Section 8.

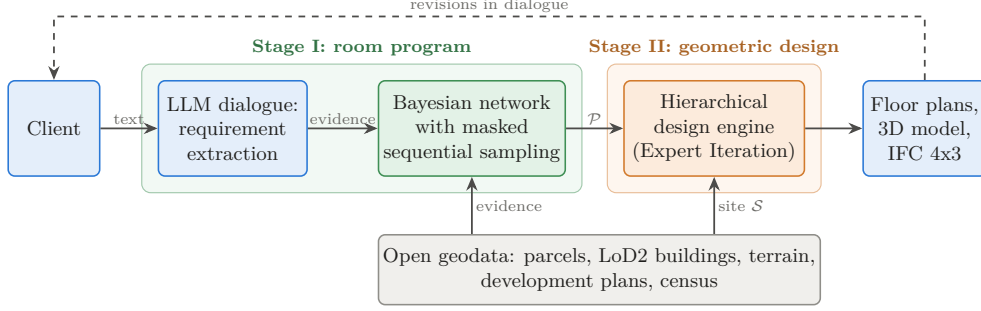


Fig. 3 System pipeline in two stages. Stage I: the LLM handles dialogue only, and the room program is sampled from a learned Bayesian network under hard constraints. Stage II: the design engine produces the geometry. Open geodata enter both stages

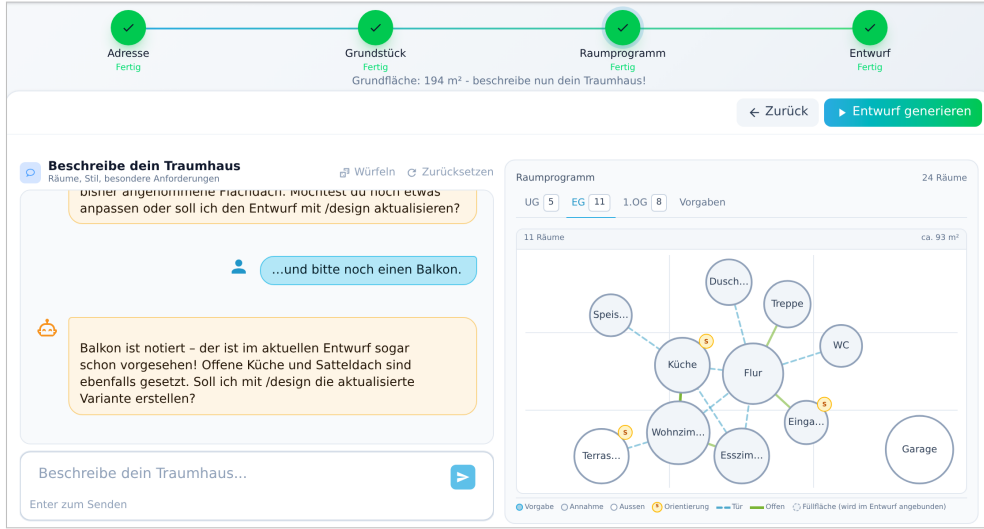


Fig. 4 Requirement elicitation: dialogue with the assistant (left) and the resulting room program as a bubble diagram of rooms and connections (right). *Raumprogramm*: room program; UG, EG, 1.OG: basement, ground floor, and first upper floor; *Vorgabe/Annahme*: specified/assumed; *Tür/Offen*: door/open; room names as in Fig. 8

6 Stage I: Learning Room Programs

6.1 Data: captured architectural designs

Because no suitable public dataset exists, we built an in-house corpus of architectural designs of German single-family houses, captured manually with a dedicated tool (Fig. 5). Each record contains building parameters (type, stories, footprint, wall height, roof form and pitch, street direction) and, per story, the rooms with type, area, and compass orientation as well as the typed connections between them. The designs originate from projects of the second author's architectural practice and were

captured by a student intern in that office. Records pass a consistency validation in the capture tool before use; the current model is trained on 168 validated designs.

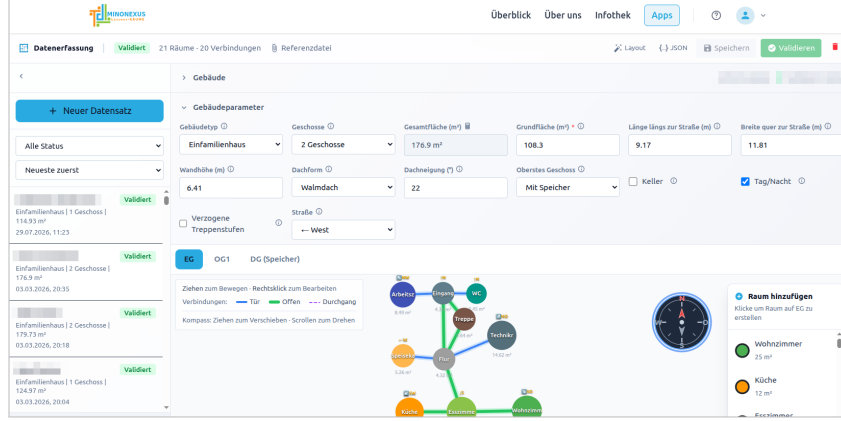


Fig. 5 Capture tool for architectural designs: building parameters (top) and the room graph of the ground floor with typed connections (door, open, passage) and orientation. *Gebäudetyp*: building type; *Geschosse*: stories; *Grundfläche*: footprint; *Dachform/Dachneigung*: roof form/pitch; *Tür/Offen/Durchgang*: door/open/passage; EG, OG1, DG: ground floor, first upper floor, and attic

6.2 Bayesian network

Following Merrell et al. (2010), we model the room program with a discrete Bayesian network (Koller & Friedman, 2009)

$$P(V_1, \dots, V_n) = \prod_{i=1}^n P(V_i \mid \text{Pa}(V_i)), \quad (3)$$

whose variables describe the building envelope (number of stories, footprint and total-area class, roof type and pitch, basement, top-floor type, street direction), the existence and count of rooms per room type, per-instance story, area class and compass orientation, and the type of connection between pairs of rooms. Context variables (dominant roof form and story count of the surroundings, existence of a development plan) are parents of the corresponding envelope variables. Given the small corpus, the structure is defined by domain experts rather than learned (“domain knowledge over data”); conditional probability tables are estimated by maximum likelihood, room instances are only modeled where supported by the data, and selected marginals are calibrated against statistics of the German building stock derived from cadastral and LoD2 data. The resulting network has 23 room types and about 430 nodes. We perform inference by variable elimination using pgmpy (Ankan & Panda, 2015).

6.3 Masked sequential sampling

Forward sampling from the network frequently yields inconsistent programs, for example stories whose summed room areas exceed the available footprint. We therefore generate a program as a sequence of decisions $d_1, \dots, d_T$ in a fixed phase order—envelope, room counts, story assignment, area classes, adjacencies, connection types, orientation—and separate hard rules from soft statistics (“mask $\neq$ prior”):

$$P(d_t = a \mid d_{<t}, e) \propto \underbrace{P_{\text{BN}}(a \mid d_{<t}, e)}_{\text{soft prior}} \cdot \underbrace{m_t(a \mid d_{<t}, \mathcal{S})}_{\text{hard mask} \in \{0,1\}}, \quad (4)$$

where e is the evidence from dialogue and site. Masks implement look-ahead checks that rule out dead ends, notably a per-story area budget that reserves the minimum areas of rooms not yet placed, and a story-aware reachability check ensuring that every room remains connected to the entrance. Each candidate hard rule was tested against the captured designs; rules violated by real designs were demoted to soft priors. User requirements restrict individual decision slots; a contradiction between requirements and hard rules leads to an empty mask and an explicit rejection rather than a silent fallback. The masks ensure that a program is consistent, not that it can be built: whether the rooms fit on a story depends on the geometry, in particular on the circulation area of hallways and staircase, which is only fixed in Stage II and which the area budget can merely estimate. Stage II therefore adapts the program within limits (Section 8.7).

Compared with forward sampling, the masked sampler eliminates over-filled stories and reproduces requested room counts exactly, at a cost of a few hundred milliseconds per program. Fig. 6 illustrates one decision step and the effect on the generated programs.

Because the procedure is a sequential decision process with legal actions and a policy, it is also the substrate on which the design engine of Section 8 operates. Sampling programs from this model provides the distribution of synthetic design requests used for self-play training. This completes Stage I; the following two sections turn to the geometric design of Stage II.

7 Stage II: Approaches We Evaluated for Layout Generation

Given a consistent room program from Stage I, what remains is the geometric design problem (2). Three premises guided our search for a design engine to solve it: the process should be hierarchical, mirroring professional practice; floor plans are essentially determined by the position and configuration of walls and wall segments; and the design space mixes continuous and discrete parameters. Before arriving at the approach of Section 8, we evaluated three families of methods.

(a) one decision step (illustrative)

(b) consistency of programs

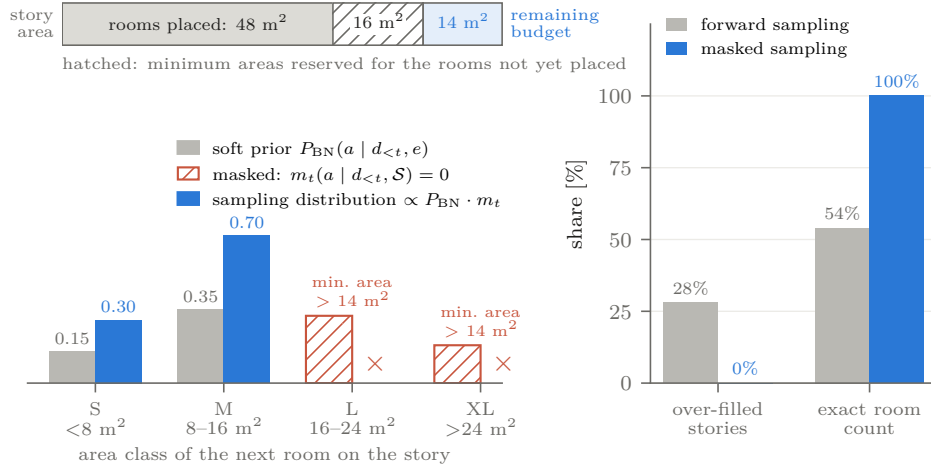


Fig. 6 Masked sequential sampling. (a) One decision step with illustrative values: the Bayesian network supplies a soft prior over the area class of the next room; the hard mask removes the classes that would exceed the remaining area budget of the story, which reserves the minimum areas of the rooms not yet placed; the product is renormalized and sampled. (b) Share of over-filled stories and of programs with exactly the requested room counts for plain forward sampling and for masked sampling

Stochastic simulation

We implemented Metropolis sampling (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953) with wall-sliding and room-assignment primitives. In preliminary experiments with suitable primitives, a basic configuration could be derived in about two minutes on a single core (AMD Threadripper, 4 GHz) and was valid in roughly 70% of the cases. However, the method is not trainable—it does not improve with a growing number of design examples—and we saw no realistic path to further improvement or to richer criteria such as orientation.

Mixed-integer optimization

Because of the mixed parameter space we formulated the layout problem as a mixed-integer (non-)linear program and solved it with the open-source solver Bonmin (COIN-OR, 2026). The complexity of formulating the design rules as algebraic constraints and objective terms grew quickly. Non-overlap, adjacency, and reachability require many binary auxiliary variables and big-M constraints, while soft criteria such as privacy gradients or orientation are hard to express in algebraic form. Rooms were restricted to rectangles, since flexible room geometries could not be modeled. Solve times increased steeply with the number of rooms, and multi-story problems with a shared staircase could not be solved in acceptable time. Moreover, like stochastic search, the approach does not learn from data, and every new design rule requires reformulating the model.

Deep reinforcement learning

Viewing design as a sequence of decisions naturally suggests reinforcement learning (RL; Sutton & Barto, 2018). We experimented with proximal policy optimization (PPO; Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) in TorchRL (PyTorch Contributors, 2026) and Stable-Baselines3 (Raffin et al., 2021), extended by a custom hierarchy of levels in which decisions mask the action spaces of lower levels. We found that (i) hierarchical masking across levels was difficult to realize and train; (ii) a pure tensor representation of variable structures—number of rooms, number of wall segments—leads to large, unwieldy observation and action spaces; and (iii) debugging reward shaping was very hard and, given the sub-symbolic representation, hardly interpretable. Per-option success rates eventually dropped below 50% due to distribution shift between levels. The difficulties described by Irpan (2018) are amplified in a hierarchical multi-level setting.

Consequence

Irpan (2018) also notes that, where a ground-truth model of the environment is available, MCTS often outperforms model-free RL. Our design simulator evaluates any reached state exactly against the cost function Φ and the validity constraints, i.e., such a model exists. We therefore built on MCTS and, specifically, on the AlphaZero paradigm (Silver et al., 2017), in which a system learns Go to mastery purely by self-play. Two differences to board games shaped our design:

1. In Go, every move occupies one point of a discrete grid. Design actions live in a mixed parameter space and differ between design phases. Our orientation on real architectural practice (one author is an architect with more than 20 years of professional experience) convinced us that design primitives are more complex and, above all, must be organized hierarchically.
2. There is no opponent whose moves must be anticipated. The problem is a single-agent planning problem with an exact simulator, which allows search and evaluation strategies that are not available in adversarial games.

8 Stage II: Hierarchical Design Engine

8.1 Design process as a hierarchy of options

Professional residential design proceeds roughly as follows: (1) analyze the site and derive the possible building envelope (buildable area, plot shape, number of stories); (2) derive the room program including the target topology; (3) decide on circulation (access, staircase), which strongly determines the design; (4) zone the floors, e.g., into living and sleeping areas; (5) define individual rooms; and (6) refine, including checks against standard furniture and use. Steps (1) and (2) are covered by Sections 4 and 6. Steps (3)–(6) are modeled as *options* $o = (\mathcal{I}_o, \pi_o, \beta_o)$ with initiation set $\mathcal{I}_o$, intra-option policy π_o , and termination condition β_o (Sutton et al., 1999).

A top-level *selector* chooses at each macro step a pair (o, ι) of option type and instance—e.g., a particular room, a cluster of rooms, a story, or a wall locus. A phase machine restricts which option types are admissible (Fig. 7):

- **Circulation core / staircase.** Carves the circulation core from the building outline and commits a staircase column. Candidate circulation archetypes are enumerated; an alternative option places the staircase directly from a discrete set of types and entry configurations.
- **Zoning.** Splits a functionally ambiguous cluster into zone-pure sub-clusters, e.g., living versus sleeping.
- **Room definition.** Resolves a cluster into typed rooms by successive subdivision.
- **Refinement.** Wall-level operations that resize, re-orient or re-proportion rooms, establish adjacencies, open walls, widen rooms, and lay out corridors.
- **Polish / stop.** After the first valid state is reached, the selector may stop or continue to improve; the best valid state found is kept.

The design space itself remains continuous: wall positions and room dimensions are real-valued. The learned policies, however, choose from a finite set of candidate actions at each decision. Continuous parameters such as wall offsets or corridor widths are offered as a few bins, defined either as fixed steps or relative to the current state. Relative bins, for example proportional to the area being divided, adapt the resolution to the size of the element. Given the discrete choice, the exact geometry follows deterministically; wall snapping aligns new walls with nearby existing ones, also across stories (Section 8.6), but there is no fixed grid. Search and learning thus operate over discrete actions, as in AlphaZero, while the geometry remains continuous. The selector network handles a variable number of instances by encoding each instance separately and scoring it against a pooled state representation.

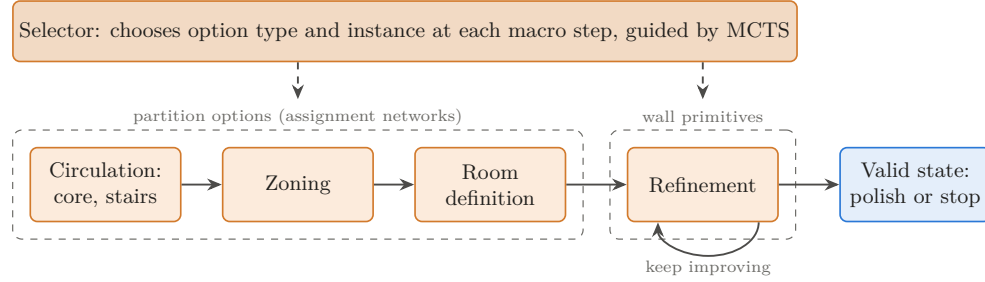


Fig. 7 Option hierarchy of the design engine. A phase machine gates the admissible options; the selector chooses option type and instance at each macro step. Each option has its own learned policy

Fig. 8 shows intermediate states of one design run. The stories are drawn apart and the exterior walls transparent, so that the interior layout of every story remains visible.

8.2 Assignment networks

The partition options—circulation, zoning, and room definition—share a recurring sub-problem: distributing a set of room-type slots of the program over a variable number of geometric zones such that the resulting cost is minimal. We solve it with *assignment networks*, pointer-style networks that encode slots (room type, target

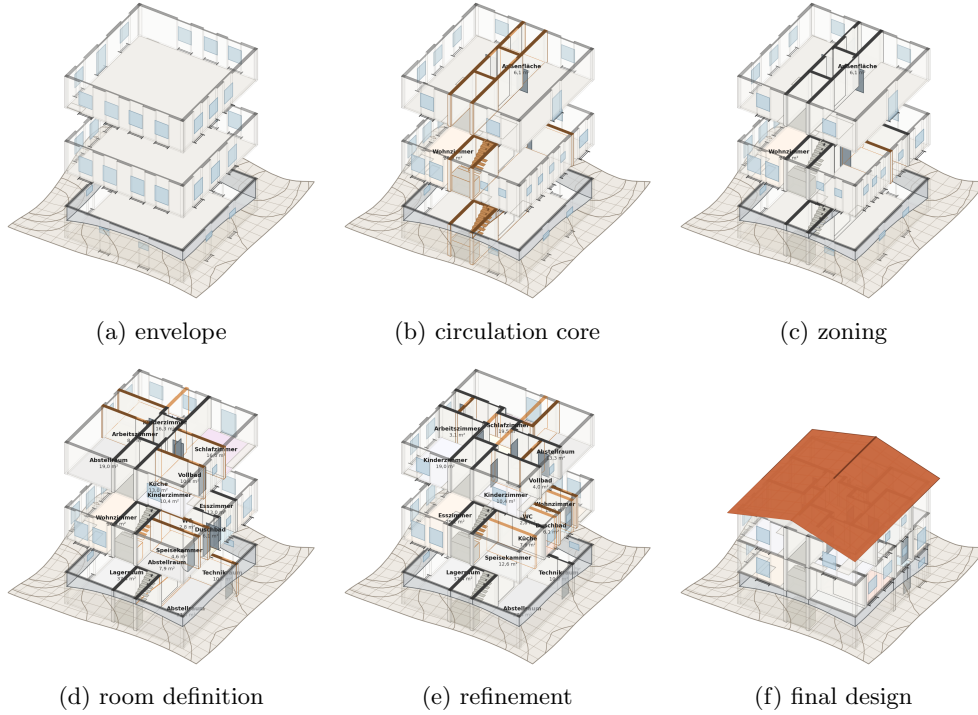


Fig. 8 Intermediate states of one design run for a two-story house with basement, rendered from the recorded trajectory: (a) building envelope, (b) circulation core and staircase, (c) zoning, (d) all rooms defined, (e) after wall-level refinement, (f) final design after area adjustment and detailing, stories collapsed. The translucent surface indicates the terrain schematically; the basement lies below it. Exterior walls are transparent, stories in (a)–(e) are drawn apart and shown without the roof; elements added by a step are highlighted in orange; doors are drawn as open leaves in gray. Room labels are in German, e.g., *Wohnzimmer*: living room; *Küche*: kitchen; *Esszimmer*: dining room; *Schlafzimmer*: bedroom; *Kinderzimmer*: child’s room; *Arbeitszimmer*: study; *Vollbad/Duschbad*: full bathroom/shower room; *Abstellraum*: storage room; *Lagerraum*: storeroom; *Technikraum*: utility room

area, required connections, privacy level, orientation preference) and zones (geometry, position, size, current occupancy, neighborhood) separately. Because they score each slot–zone pair, they are independent of the number of zones. A value head estimates the quality of the resulting partition. Each level uses its own network instance, since sharing one network across levels led to interference between their value scales. Depending on the size of the decision, assignments are taken greedily by value or refined by tree search.

8.3 Primitives

Options are built on primitives that create and manipulate wall segments: splitting and merging rooms, sliding walls and wall chains, swapping rooms, carving and widening corridors, opening walls, and straightening offsets. Unlike pure dissection, wall-sliding primitives allow non-rectangular rooms. Once the staircase is committed, its walls are locked, and any move that would invalidate the staircase is rejected.

8.4 Cost function

The cost function, whose origin lies in [Merrell et al. \(2010\)](#) and which was developed further with architectural expertise, separates hard validity constraints from weighted soft criteria. Validity requires, among others, a committed staircase aligned across all stories, no overlapping or degenerate rooms, no rooms that are unreachable, a connected circulation system per story that the staircase joins, and exterior access to every entrance. The soft cost is a weighted sum

$$\Phi(X; \mathcal{P}, \mathcal{S}) = \sum_k w_k \varphi_k(X; \mathcal{P}, \mathcal{S}), \quad (5)$$

with terms in the following groups:

- **dimensions:** deviation of area, aspect ratio, and compactness from the program and from the distributions observed in the captured designs;
- **shape:** number of edges, convexity, irregularities of rooms and of the floor outline;
- **topology:** missing required and preferred connections, weighted differently;
- **circulation:** staircase geometry and alignment, hallway area share, corridor and room bottlenecks below minimum widths, use of facade length for circulation;
- **orientation and daylight:** deviation of rooms and entrance from their preferred compass orientation, exterior wall share;
- **program coverage** and user wishes, e.g., a requested staircase type.

Soft terms can encode design knowledge that is not a strict rule. An example is the *intimacy gradient* (pattern 127 in [Alexander, Ishikawa, & Silverstein, 1977](#)), by which spaces are ordered from public to private when moving away from the entrance. It enters as a soft criterion in the evaluation of zoning decisions and through the privacy level of each room in the assignment networks, whereas a statement such as “every room must be reachable” is a hard constraint. Individual terms penalize defects in a graded way; for example, corridors narrower than a minimum width $w_{\min}$ are penalized quadratically,

$$\varphi_{\text{corr}}(X) = \sum_{c \in \mathcal{C}(X)} \max(0, w_{\min} - w_c)^2, \quad (6)$$

where $\mathcal{C}(X)$ is the set of corridor segments with width w_c . Following a recent revision, rectangular rooms, rectangular floors, and fitting staircases incur zero cost, so that Φ measures explicit defects rather than an irreducible baseline.

8.5 Expert Iteration in synthetic self-play

Since no suitable training data exist, the engine generates its own: it designs houses for synthetic design requests sampled from the room-program model of Section 6, evaluates the results with Φ and learns from the outcome (Fig. 9). We follow Expert Iteration ([Anthony et al., 2017](#)):

- **Expert.** At the selector level, an AlphaZero-style MCTS ([Silver et al., 2017](#)) with Gumbel-based root selection and sequential halving ([Danihelka, Guez, Schrittwieser, & Silver, 2022](#)) searches over option choices. Inside options, smaller searches or

learned scorers select primitive actions; the exact simulator evaluates the resulting states.

- **Apprentices.** A policy-value network for the selector, one network per option, and the assignment networks. The search provides improved policy targets π^{MCTS} ; the value target is derived from the terminal outcome,

$$z = \begin{cases} 0 & \text{if no valid state was reached,} \\ q(\Phi(X_{\text{best}})) & \text{otherwise,} \end{cases} \quad (7)$$

with a monotonically decreasing, bounded map q of the cost of the best valid state found.

- **Training.** Options are first pre-trained individually on states from a curriculum of design phases; the full hierarchy is then trained jointly in iterations of self-play episodes, replay, and network updates. Candidate networks are promoted only after paired evaluation against the current reference on a fixed set of design tasks, with rollback otherwise.

Training runs on CPU-only hardware (48 virtual cores); one iteration takes about two hours. Searches are expensive and are therefore used during training; at serving time, the learned apprentices guide a much smaller search.

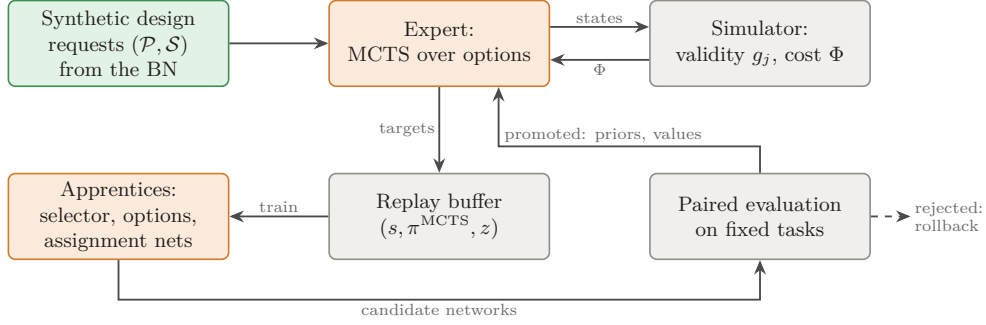


Fig. 9 Expert Iteration in synthetic self-play. The search-based expert produces improved targets for the neural apprentices, which in turn guide the search. New networks are promoted only after paired evaluation

8.6 Multi-story handling

Vertical circulation is treated as a first-class decision. A single staircase column is placed at the same absolute position on every story, determined from the ground floor and sized to fit the smallest story; upper stories may choose their own circulation archetype around it. Different staircase types (straight, quarter-turn, half-turn) are realized as distinct variants with their actual entry and exit edges. Staircase alignment is a hard constraint, and cross-story coupling terms enter the cost. Wall snapping also acts across stories: new walls are aligned with the walls of the stories above and

below where possible, which favors walls that stack vertically and thus a plausible load-bearing structure.

8.7 Adapting the program

Stage I fixes the program before any geometry exists. It can therefore neither guarantee that the program can be built on the given footprint nor predict its space requirement exactly. The largest uncertainty lies in the circulation area, which depends on the circulation archetype, the staircase type, and the position of the core, all of which are decided only in Stage II. The engine therefore treats part of the program as negotiable. A room is optional if its type is optional, such as a storage room, or if it was proposed by the room-program model without hard evidence from the user’s input; rooms explicitly requested by the user are always required. If a zone turns out too small for its rooms, optional rooms can be dropped; if space remains, it can be filled with optional rooms rather than by enlarging other rooms beyond their target areas or leaving undefined residual spaces. These decisions do not follow a fixed rule for the whole building. They are taken locally within each zone, as part of the assignment and room-definition decisions (Section 8.2), and are thus weighed against the cost function. Remaining space can also be given back to the envelope: the footprint from Stage I is treated as a maximum extent, so the engine may reduce the footprint on the plot or set back an upper story, for example to form a balcony.

9 Results

9.1 Deployed system

MinoSketch is available as a browser-based service at <https://minonexus.com> and is free of charge for private clients; use requires a free account, and the number of design runs per account is limited. Designs are currently supported for plots in Germany. A user proceeds in four steps. (1) They enter the address; the parcel and the applicable regulations are then loaded, from a development plan or, if none exists, from the surroundings. (2) They inspect terrain and surroundings and describe the desired house in natural language. (3) They start the design, which proceeds from envelope and roof to walls, windows, staircase, and furniture. (4) They review floor plans with areas per story and a 3D model between the real neighboring buildings (Fig. 10). A recorded run is available as a video (Minonexus GmbH, 2026a). Basic editing functions (swapping rooms, moving wall segments with areas updated) are available, and the model can be exported as IFC 4x3 including terrain and georeferencing for further processing in common architectural CAD systems. A design is generated asynchronously in about five minutes or slightly longer.

9.2 Quantitative evaluation

Validity and quality. On a fixed set of 60 design tasks (room programs on plots), the current engine produces valid designs for 58–59 tasks (97–98%). Over the last ten training iterations, quality on this set remained stable with a slight improvement: the median cost Φ of the delivered designs moved from 114 to between 101 and 109. Since



Fig. 10 Generated design. Left: upper-floor plan with furniture, dimensions, and room areas; room names as in Fig. 8, *Familienbad*: family bathroom. Right: 3D model placed between the neighboring buildings from open LoD2 data

Φ is an internal cost scale, relative changes and paired comparisons per task are more meaningful than absolute values.

Learning in self-play. Fig. 11 shows the share of self-play episodes that reach a valid design over 72 training iterations; it rises from about 63% (mean of the first five iterations) to about 95% (mean of the last five). The curve spans several resumed training runs, partly with code changes in between, so the improvement reflects both learning and engineering.



Fig. 11 Share of self-play episodes reaching a valid design per training iteration (dots) and moving average over five iterations (line). Dashed lines mark resumptions of training, partly with code changes, so the improvement reflects both learning and engineering

Room programs. For the room-program model, masked sequential sampling produced no over-filled stories and exact room counts, compared with over-filling in about 28% of stories and correct room counts in 54% of cases for plain forward sampling from the Bayesian network.

10 Discussion and Outlook

Reception

Feedback from prospective homeowners has been very positive. Tools of this kind are perceived as empowering: clients can explore designs based on their own wishes on their own plot before meeting an architect or a construction company. Reactions from architects are mixed. Some fear replacement and question the quality of “mathematical” derivations compared with human creativity. Since “architect” is a legally protected title in Germany, we do not use it for the system; nevertheless, it has been questioned whether “architectural design” is an appropriate term for the output of an AI system. Other architects regard such systems as an opportunity: a preliminary design that manifests the client’s wishes is a robust basis for the conversation. In Germany, building applications may in principle only be submitted by authorized design professionals, predominantly architects, so there is no automatism by which a house is built as generated. Moreover, any built house requires detailed design (construction method, energy performance assessment, building services). We see MinoSketch as a starting point for, not a replacement of, professional design—including as data input to the architect’s workflow via IFC.

Low-threshold access

A design goal of MinoSketch was robustness beyond a research prototype: the system is operated as a public service for arbitrary plots and programs within its scope (Section 9). We see such low-threshold access as a possible driver for the use of AI in architectural design: it meets clients’ wish to explore their building project on their own, with the support of AI, and clients can enter the collaboration with the designer earlier and with concrete ideas of their own. Within the profession, adoption is still cautious: in a 2025 survey of German architects, 29% used AI tools in their offices, mostly on a trial basis, and AI hardly played a role in creative work ([Bundesarchitektenkammer, 2025](#)). Tools for architects are, moreover, largely developed as extensions of established CAD and BIM platforms rather than conceived as AI-native systems.

Limitations

The system currently covers detached single-family houses with one or two full stories, optionally with a basement and a converted attic story, on plots in Germany. The room-program model is learned from a small corpus, and the cost function encodes a specific set of design criteria with fixed weights. Daylight is approximated by orientation and exterior-wall terms rather than simulated. Our quantitative evaluation measures the cost function the engine is trained on; an independent assessment by architects is pending.

Next steps

- **Editing.** Beyond the existing editing functions, we are investigating which further editing options are useful for clients, without re-creating a CAD system; professional editing is possible via IFC export.

- **Feedback.** We are establishing structured feedback on generated designs along several assessment axes, initially by the architects in our team, to tune the cost function; real designs created in the system will complement synthetic self-play as a learning source.
- **Further criteria.** Integration of building services, thermal protection, and acoustic design are candidates for additional cost terms. We believe that an explicit, tool-based approach offers significant leverage for mastering this complexity and analyzing variants.

Open data

Independently of our own development, data-driven embedding of designs into their context depends on progress in the digitalization of regulations and environmental data. In Germany’s federal structure, scope and form of provision vary considerably between states, and there is room for a more uniform approach and stronger political steering ([Minonexus GmbH, 2026b, 2026c](#)). Every municipality that publishes its development plans in machine-readable form directly improves the designs generated for its area.

11 Conclusion

We presented MinoSketch, a deployed system for site-specific generative design of multi-story residential buildings. It separates dialogue (LLM), requirement modeling (Bayesian network with masked sequential sampling) and geometric design (hierarchical options trained by Expert Iteration), and it grounds every design in open geodata about the plot and its surroundings. Experience with stochastic search, mixed-integer optimization, and model-free reinforcement learning led us to a search-based, hierarchical, and trainable approach that improves through self-play in the absence of public training data. The outputs are editable building models rather than images, and they enter professional workflows via IFC 4x3.

Acknowledgements. We thank the surveying and planning authorities of the German federal states and the Federal Agency for Cartography and Geodesy (BKG) for providing open geodata.

Statements and Declarations

- **Funding:** This work was funded by Minonexus GmbH. No external funding was received.
- **Competing interests:** The authors are founders and employees of Minonexus GmbH, which develops and operates the MinoSketch service described in this article.
- **Ethics approval and consent to participate:** Not applicable.
- **Consent for publication:** Not applicable.
- **Data availability:** The corpus of captured architectural designs and the trained models are proprietary and not publicly available. The geodata used are openly available from the surveying and planning authorities of the German federal states under the licenses stated by the respective providers (e.g., dl-de/by-2-0,

CC BY 4.0). Sources of the geodata shown in the figures: © GeoBasis-DE/BKG (2026); Landesamt für Geoinformation und Landentwicklung Baden-Württemberg (LGL), www.lgl-bw.de; Freie und Hansestadt Hamburg, LGV.

- **Materials availability:** Not applicable.
- **Code availability:** The source code and trained models are proprietary, as MinoSketch is developed and operated commercially by Minonex GmbH, and are not publicly available. The deployed system can be used free of charge by private clients at <https://minonex.com>.
- **Author contributions:** M.H.: conceptualization, methodology, software, investigation, writing – original draft. H.H.: conceptualization, architectural domain model (design process, cost criteria), resources (design corpus from her architectural practice), supervision of data capture, validation of generated designs, writing – review and editing. H.H. holds a Dipl.-Ing. degree in architecture from TU Dresden and is a registered architect (Bavarian Chamber of Architects) with more than 20 years of professional experience; M.H. holds a Dr.-Ing. degree. All authors read and approved the final manuscript.
- **Use of generative AI:** A large language model (Claude, Anthropic) was used to assist in editing and translating the manuscript from the authors’ notes and internal technical documentation. The authors reviewed and revised all content and take full responsibility for it.

References

- Alexander, C., Ishikawa, S., Silverstein, M. (1977). *A Pattern Language: Towns, Buildings, Construction*. New York, NY: Oxford University Press.
- Ankan, A., & Panda, A. (2015). pgmpy: Probabilistic Graphical Models using Python. *Proceedings of the 14th Python in Science Conference (SciPy 2015)* (pp. 6–11).
- Anthony, T., Tian, Z., Barber, D. (2017). Thinking Fast and Slow with Deep Learning and Tree Search. *Advances in Neural Information Processing Systems 30 (NIPS 2017)*.
- Autodesk (n.d.). *Autodesk Forma*. Retrieved 2026-10-02, from <https://www.autodesk.com/de/products/forma/overview>
- Bundesarchitektenkammer (2025, November). *Architektenbefragung 2025: KI, Teilzeit und Bürokratie prägen den Berufsalltag*. Retrieved 2026-10-02, from <https://bak.de/presse/pressemitteilungen/architektenbefragung-2025-ki-teilzeit-und-buerokratie-praegen-den-berufsalltag/>
- COIN-OR (2026). *Bonmin Home Page*. Retrieved 2026-10-01, from <https://www.coin-or.org/Bonmin/>

- Danihelka, I., Guez, A., Schrittwieser, J., Silver, D. (2022). Policy improvement by planning with Gumbel. *International Conference on Learning Representations (ICLR)*.
- Das, S., Day, C., Hauck, A., Haymaker, J., Davis, D. (2016). Space Plan Generator: Rapid Generation & Evaluation of Floor Plan Design Options to Inform Decision Making. *ACADIA 2016: Post-Human Frontiers* (pp. 106–115). Ann Arbor (Michigan), USA. Retrieved 2026-10-01, from http://papers.cumincad.org/cgi-bin/works/paper/acadia16_106
- Finch3D (n.d.). *Finch: Optimizing architecture*. Retrieved 2026-10-02, from <https://www.finch3d.com>
- Flemming, U. (1978, December). Wall Representations of Rectangular Dissections and Their Use in Automated Space Allocation. *Environment and Planning B: Planning and Design*, 5(2), 215–232, <https://doi.org/10.1068/b050215> Retrieved 2026-10-01, from <https://journals.sagepub.com/doi/10.1068/b050215>
- Hu, R., Huang, Z., Tang, Y., Van Kaick, O., Zhang, H., Huang, H. (2020, August). Graph2Plan: Learning floorplan generation from layout graphs. *ACM Transactions on Graphics*, 39(4), 118:1–118:14, <https://doi.org/10.1145/3386569.3392391> Retrieved 2026-10-01, from <https://dl.acm.org/doi/10.1145/3386569.3392391>
- Irpan, A. (2018, February). *Deep Reinforcement Learning Doesn't Work Yet* [Blog Post]. Retrieved 2026-10-01, from <https://www.alexirpan.com/2018/02/14/rl-hard.html>
- ISO (2024). *ISO 16739-1:2024 Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries – Part 1: Data schema*. Retrieved 2026-10-02, from <https://www.iso.org/standard/84123.html>
- Kalervo, A., Ylioinas, J., Häikiö, M., Karhu, A., Kannala, J. (2019). CubiCasa5K: A Dataset and an Improved Multi-task Model for Floorplan Image Analysis. M. Felsberg, P.-E. Forssén, I.-M. Sintorn, & J. Unger (Eds.), *Image Analysis* (Vol. 11482, pp. 28–40). Cham: Springer International Publishing. Retrieved 2026-10-01, from https://doi.org/10.1007/978-3-030-20205-7_3
- Kocsis, L., & Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. J. Fürnkranz, T. Scheffer, & M. Spiliopoulou (Eds.), *Machine Learning: ECML 2006* (Vol. 4212, pp. 282–293). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved 2026-10-01, from https://doi.org/10.1007/11871842_29
- Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA: MIT Press.

- Li, C., Zhang, T., Du, X., Zhang, Y., Xie, H. (2024). *Generative AI for Architectural Design: A Literature Review*. Retrieved 2026-09-30, from <https://arxiv.org/abs/2404.01335>
- Li, Z., Li, S., Hinchcliffe, G., Maitless, N., Birbilis, N. (2024, June). *Automated architectural space layout planning using a physics-inspired generative design framework* (No. arXiv:2406.14840). arXiv. Retrieved 2026-10-02, from <http://arxiv.org/abs/2406.14840>
- Lobos, D., & Donath, D. (2010, December). The problem of space layout in architecture: A survey and reflections. *Arquitetura Revista*, 6(2), 136–161, <https://doi.org/10.4013/arq.2010.62.05> Retrieved 2026-10-01, from <http://revistas.unisinos.br/index.php/arquitetura/article/view/4554>
- Merrell, P., Schkufza, E., Koltun, V. (2010, December). Computer-generated residential building layouts. *ACM Transactions on Graphics*, 29(6), 1–12, <https://doi.org/10.1145/1882261.1866203> Retrieved 2026-10-01, from <https://dl.acm.org/doi/10.1145/1882261.1866203>
- Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E. (1953, June). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6), 1087–1092, <https://doi.org/10.1063/1.1699114> Retrieved 2026-10-03, from <https://doi.org/10.1063/1.1699114>
- Minonex GmbH (2026a). *Architekturentwurf mit MinoSketch*. Retrieved 2026-09-30, from <https://minonex.com/minosketch/film-ablauf> (Video (German))
- Minonex GmbH (2026b). *Flurstücksdaten in Bayern: Die Antwort der Staatsregierung*. Retrieved 2026-10-01, from <https://minonex.com/minonex/flurstuecksdaten-bayern>
- Minonex GmbH (2026c). *XPlanung beim Bund: Was das Ministerium nicht weiß*. Retrieved 2026-10-01, from <https://minonex.com/minonex/xplanung-ifg-bund>
- Nauata, N., Chang, K.-H., Cheng, C.-Y., Mori, G., Furukawa, Y. (2020). House-GAN: Relational Generative Adversarial Networks for Graph-Constrained House Layout Generation. A. Vedaldi, H. Bischof, T. Brox, & J.-M. Frahm (Eds.), *Computer Vision – ECCV 2020* (Vol. 12346, pp. 162–177). Cham: Springer International Publishing. Retrieved 2026-10-01, from https://doi.org/10.1007/978-3-030-58452-8_10

- Nisztuk, M., & Myszkowski, P.B. (2019, September). Hybrid Evolutionary Algorithm applied to Automated Floor Plan Generation. *International Journal of Architectural Computing*, 17(3), 260–283, <https://doi.org/10.1177/1478077119832982> Retrieved 2026-10-01, from <https://journals.sagepub.com/doi/10.1177/1478077119832982>
- PyTorch Contributors (2026). *TorchRL — torchrl 0.14 documentation*. Retrieved 2026-10-01, from <https://docs.pytorch.org/rl/stable/index.html>
- Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268), 1–8, Retrieved from <https://jmlr.org/papers/v22/20-1364.html>
- Robert McNeel & Associates (n.d.-a). *Grasshopper*. Retrieved 2026-10-02, from <https://www.grasshopper3d.com/>
- Robert McNeel & Associates (n.d.-b). *Rhinoceros 8*. Retrieved 2026-10-02, from <https://www.rhino3d.com/>
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017, August). *Proximal Policy Optimization Algorithms* (No. arXiv:1707.06347). arXiv. Retrieved 2026-10-01, from <https://doi.org/10.48550/arXiv.1707.06347>
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Hassabis, D. (2017, October). Mastering the game of Go without human knowledge. *Nature*, 550(7676), 354–359, <https://doi.org/10.1038/nature24270>
- Sutton, R.S., & Barto, A.G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). Cambridge, MA: MIT Press.
- Sutton, R.S., Precup, D., Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2), 181–211, [https://doi.org/10.1016/S0004-3702\(99\)00052-1](https://doi.org/10.1016/S0004-3702(99)00052-1)
- syte GmbH (n.d.). *syte: Die KI-Plattform für Grundstücke mit Potenzial*. Retrieved 2026-10-02, from <https://www.syte.ms/>
- van Engelenburg, C., Mostafavi, F., Kuhn, E., Jeon, Y., Franzen, M., Standfest, M., ... Khademi, S. (2024). MSD: A Benchmark Dataset for Floor Plan Generation of Building Complexes. *European Conference on Computer Vision (ECCV)*.
- Wu, W., Fu, X.-M., Tang, R., Wang, Y., Qi, Y.-H., Liu, L. (2019). Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics*, 38(6), 234:1–234:12, <https://doi.org/10.1145/3355089.3356556>

- XLeitstelle (n.d.). *Was ist XPlanung?* Retrieved 2026-10-04, from https://xleitstelle.de/xplanung/ueber_xplanung
- Zawidzki, M., & Szklarski, J. (2020, March). Multi-objective optimization of the floor plan of a single story family house considering position and orientation. *Advances in Engineering Software*, 141, 102766, <https://doi.org/10.1016/j.advengsoft.2019.102766> Retrieved 2026-10-01, from <https://linkinghub.elsevier.com/retrieve/pii/S0965997819301310>
- Zhang, J. (2025, April). Automated Parametric Generation of Residential Floor Plans Using Grasshopper and Python. *Journal of Research in Science and Engineering*, 7, 72–77, [https://doi.org/10.53469/jrse.2025.07\(4\).15](https://doi.org/10.53469/jrse.2025.07(4).15)