RESEARCH-ARTICLE

# Tesseract: Unfolding Navigable Graph Representations from Low-Semantic Floor Plans

**YAQOOB ANSARI**, Carnegie Mellon University, Pittsburgh, PA, United States

**AMMAR KARKOUR**, Carnegie Mellon University, Pittsburgh, PA, United States

**EDUARDO FEO FLUSHING**, Carnegie Mellon University, Pittsburgh, PA, United States

**KHALED A HARRAS**, Carnegie Mellon University, Pittsburgh, PA, United States

Open Access Support provided by:

Carnegie Mellon University

**PDF Download**
**3748636.3762771.pdf**
**20 January 2026**
**Total Citations:** 0
**Total Downloads:** 160

**Published:** 03 November 2025

**Citation in BibTeX format**

SIGSPATIAL '25: 33rd ACM International Conference on Advances in Geographic Information Systems
*November 3 - 6, 2025*
*MN, Minneapolis, USA*

**Conference Sponsors:**
SIGSPATIAL

# Tesseract: Unfolding Navigable Graph Representations from Low-Semantic Floor Plans

Yaqoob Ansari
Carnegie Mellon University
Pittsburgh, USA
yansari@andrew.cmu.edu

Ammar Karkour
Carnegie Mellon University
Pittsburgh, USA
akarkour@andrew.cmu.edu

Eduardo Feo Flushing
Carnegie Mellon University
Pittsburgh, USA
efeoflus@andrew.cmu.edu

Khaled A. Harras
Carnegie Mellon University
Pittsburgh, USA
kharras@andrew.cmu.edu

## Abstract

Indoor maps are essential for navigation, resource allocation, and autonomous operation in complex environments, yet creating them at scale has long been impeded by high costs and specialized hardware requirements. We present *Tesseract*, a modular system that transforms ordinary low-semantic floor plan images into navigable graph structures, without requiring specialized sensors or 3D modeling tools. Through *Tesseract*, we integrate deep learning modules for text detection and door classification. We then implement a novel floodfill-based segmentation and graph optimization solution. *Tesseract* ultimately generates semantically rich, compact graph representations of the original floor plans that are computationally parsable for indoor navigation applications. We evaluate *Tesseract* across two large-scale university buildings as well as a benchmark dataset, demonstrating high navigational completeness despite variations in layout complexity. The system processes floor plans efficiently, with runtime scaling linearly to the number of detected regions, thus remaining practical for large-scale deployments. Graph pruning reduces the initially dense connectivity—typically quadratic in the number of regions—to a sparse structure, yielding up to 78% fewer nodes and 70% fewer edges, all without compromising connectivity. Moreover, geometric fidelity is preserved within 80–86% of true real-world distances. These findings establish *Tesseract* as a robust and scalable solution, broadening access to automated indoor navigation and spatial analytics.

## CCS Concepts

• **Information systems → Geographic information systems**; • **Computing methodologies → Image-based rendering**; • **Theory of computation → Graph algorithms analysis**.

## Keywords

Indoor Spatial Representation, Graph-based Navigation, Semantic Map Extraction, Indoor Navigation

**ACM Reference Format:**
Yaqoob Ansari, Ammar Karkour, Eduardo Feo Flushing, and Khaled A. Harras. 2025. Tesseract: Unfolding Navigable Graph Representations from Low-Semantic Floor Plans. In *The 33rd ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '25), November 3–6, 2025, Minneapolis, MN, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3748636.3762771

## 1 Introduction

Understanding and utilizing indoor spaces effectively requires more than visual representations. It demands structured, machine-readable models that can support reasoning, navigation, and autonomous operation [27, 30]. While outdoor environments have benefited from advances in satellite imagery and large-scale geospatial annotation [7, 40], comparable progress in indoor environments has been limited. Indoor spatial data remains fragmented, and its extraction often depends on manual modeling or specialized equipment. Nevertheless, high-quality indoor models are essential across a range of domains, including autonomous systems [13], robotics [44], augmented reality [38], intelligent building management [48], and indoor localization [17, 29]. These applications require spatial representations that are not only geometrically accurate but also semantically structured to enable reliable navigation [16, 19] task execution [21], and system integration [14, 39].

Such models can support efficient path planning for humans and mobile agents [9, 28, 37], improve the performance of indoor localization systems [18, 29], and facilitate more intuitive interactions within human-centered spaces. However, detailed geometry alone is insufficient. Many real-world applications require knowledge of spatial connectivity, functional labeling, and traversability. These properties are best captured through structured graph-based models, which abstract spatial layouts into discrete regions connected by defined transitions. To fully realize the potential of indoor spatial systems, there is a need to move beyond conventional map generation and toward the creation of computationally parsable navigable graphs. These graphs provide a compact and interpretable foundation for machine reasoning, encoding semantic regions, adjacency relations, and accessibility constraints in a form that is suitable for large-scale automation and decision-making.

In this paper, to address these challenges, we propose *Tesseract*, a novel system that generates navigable indoor graphs directly from low-semantic images. *Tesseract* leverages deep learning for text detection, semantic inference, and door identification with our novel floodfill-based segmentation algorithm and graph construction techniques. Our floodfill-based methodology employs a radial-based seeding strategy, initiating multiple floodfill operations from detected text labels used as space or room identifiers. As such, we precisely delineate distinct indoor regions and robustly overcome ambiguities posed by text boundaries or room artifacts in noisy, low-semantic floor plan imagery. The resulting output is a structured graph representation that encodes both the geometric layout and semantic content of indoor environments. Our modular architecture enables targeted deployment of machine learning models on well-bounded subtasks, allowing for straightforward model updates and compatibility with advances in computer vision. This design ensures that the system remains both flexible and adaptable while retaining interpretability and control over each stage of the pipeline.

We build *Tesseract* to be a truly deployable system that addresses concerns of privacy, practicality, and scale. It safeguards privacy by processing offline, pre-existing floor plans without relying on sensitive user or real-time data. This makes it well-suited for deployment in secure institutional environments where data protection is paramount. *Tesseract* only uses basic, low-semantic floor plan images, commonly found in fire evacuation guides, information kiosks in malls and airports, or architectural records. Although modern CAD formats (e.g., DXF, DWF) offer detailed vector representations, with tools that can extract structured primitives from them, such files are often unavailable, or may include sensitive architectural metadata or proprietary design layers, making them unsuitable for broad distribution due to privacy and ownership concerns; in contrast, floor plan images are more abundant, publicly accessible, and easier to process at scale. Finally, *Tesseract* avoids specialized hardware or complicated workflows, enabling rapid navigable indoor graph generation even for non-expert users. By addressing the cost, complexity, and privacy barriers that have traditionally hindered large-scale indoor modeling, *Tesseract* offers a practical and scalable approach for producing structured, machine-usable spatial representations from widely available visual inputs.

We comprehensively evaluate *Tesseract* across two large-scale university buildings as well with the SESYD [11] benchmark. Our evaluation spans four key dimensions: navigational completeness, exit reachability, computational efficiency, and geometric fidelity. Our results show that *Tesseract* consistently produces highly connected graphs, with average navigational completeness scores higher than 90%, and reaching 98%. Exit reachability remains above 85% across all datasets. In terms of efficiency, the system maintains a total runtime of under two minutes per image patch, even on large and densely annotated floor plans. Our efficient solution, via structural pruning, achieves significant compression—up to 78% reduction in nodes and 74% in edges—without compromising connectivity. Moreover, geometric fidelity remains high, with over 90% alignment between graph-based and true Euclidean distances across room pairs. These results highlight *Tesseract*'s robustness, scalability, and suitability for generating compact, semantically rich indoor graphs from low-semantic inputs.

## 2 Related Work

This section examines the progression of indoor modeling, from robotics-driven approaches to data-centric techniques that minimize reliance on specialized hardware. We also review methods that utilize CAD drawings as input. Unlike prior approaches, our pipeline processes static floor plan images to generate structured, navigable indoor graphs—sidestepping the need for specialized sensors or CAD/BIM models while supporting scalable deployment.

In *mobile robotics*, robots rely heavily on their capacity to both perceive and map the environment [], a task accomplished through Simultaneous Localization And Mapping (SLAM) [12]. SLAM enables robots to construct spatial maps, integrate sensory data, and localize themselves for various tasks. However, these spatial maps are primarily geometric, lacking the semantic richness needed for effective interaction in human-oriented spaces; they typically omit information such as room names, office numbers, space types (e.g., lab, lounge, restroom), or the functional purpose of a region—details that are crucial for applications like context-aware navigation, emergency response, or space utilization analysis [1, 46]. This limitation has prompted the development of semantic indoor mapping techniques, which incorporate meaningful information into the mapping process. Some approaches involve human collaboration, such as engaging in conversations to enrich maps with semantic data [4], while others utilize machine learning to improve scene comprehension and region identification. Such solutions include the creation of hierarchical 3D scene graphs that define spatial relationships between rooms [20], vision-to-language models that describe scenes [6], and region classifiers that label sensor data from RGB-D inputs [41]. Although these advances in robotic mapping are promising, their reliance on complex hardware remains a significant barrier to broader adoption, limiting their practical use outside specialized environments [5, 22, 42]. Contrary to these methods, our approach removes dependence on robotics platforms and high-end sensors by leveraging widely available data sources.

To bypass the reliance on specialized sensors or robotic platforms, some approaches leverage pre-existing symbolic or textual data about buildings. These *data-driven semantic methods* focus on using structured sources such as CityGML, indoorGML [36], and Apple's IMDF. These formats are rich in semantic and topological details but require significant manual labor and in-depth expertise [25]. In response, recent developments suggest the use of natural language descriptions of building features, paired with advanced deep learning models such as Stanford Scene Graph Parser [8] or Generative Adversarial Networks (GANs) [33], to produce graph-based indoor maps. This approach minimizes dependence on intricate data formats but still demands a comprehensive understanding of the building's layout and function, making it challenging to crowdsource or delegate to general users. Additionally, these models require extensive datasets of indoor environments for training, a task complicated by privacy concerns and the inherent sensitivity of indoor space data.

Separately, a growing body of work focuses on extracting spatial representations directly from visual documents such as CAD drawings or 2D floor plan images. CAD files typically contain precise geometric attributes, enabling the reconstruction of robust 3D models that facilitate navigation and analytical tasks [31, 32]. Effective use of CAD data often necessitates specialized expertise, comprehensive preprocessing, and compliance with standardized formats—factors that can hinder large-scale deployment. As an alternative, recent approaches have focused on automatically deriving indoor spatial information directly from 2D floor plan images. For instance, walls, doors, and other structural elements can be automatically identified using deep learning techniques for segmentation and object detection to generate semantically rich indoor maps [23, 43, 47]. While these methods eliminate the need for sophisticated hardware and reduce complexity, they rely on comprehensive training datasets and careful tuning to handle floor plan style and content variations. Ongoing efforts aim to enhance the robustness of these workflows, ensuring reliable indoor mapping from diverse data sources, further lowering the barriers to adoption.

With the rapid evolution of Vision-Language Models (VLMs), it is natural to ask whether general-purpose multimodal systems—such as GPT-4V or Gemini—can perform end-to-end floor plan understanding by directly generating a structured graph from an input image. While VLMs have shown promising results on small or synthetic floor plans, their performance deteriorates on complex or large-scale layouts due to limitations in spatial precision, global consistency, and fine-grained localization[10, 24]. These models often miss small but critical features (e.g., narrow doorways) or generate inconsistent topologies that break navigational connectivity[45]. Moreover, they currently offer no guarantees of semantic alignment, graph completeness, or reproducibility—attributes that are essential for navigation and safety-critical applications[35]. Beyond performance, reliance on proprietary cloud-based APIs raises privacy concerns and complicates deployment at scale [15, 26]. In contrast, *Tesseract* is designed specifically for this task: it operates offline, supports large architectural inputs, and produces structured, high-fidelity graphs optimized for semantic accuracy and graph-theoretic integrity. While VLMs may eventually complement these systems, current limitations underscore the continued need for task-specific, transparent, and controllable systems like *Tesseract*.

## 3 *Tesseract*: A Modular Architecture

### 3.1 Architectural Overview

The proposed *Tesseract* is a modular framework designed to convert annotated floor plan images into graph-based indoor graphs suitable for downstream tasks such as navigation, spatial analytics, and simulation. The system prioritizes robustness and scalability by decomposing the mapping process into distinct processing stages, each of which can be refined or replaced independently (see Figure 1 for a schematic overview). This modular design ensures that improvements in one stage (e.g., text recognition) naturally propagate throughout the entire system without necessitating major revisions in the other components.

At a high level, *Tesseract* begins by extracting textual labels (e.g., room numbers, corridor identifiers) using a deep-learning-based text detection module. These detected labels are then refined and merged to eliminate duplication or splitting errors. Next, the system employs a floodfill-based spatial segmentation approach to delineate each labeled region (rooms, corridors, and outdoor spaces) in the image. From these segmented regions, an initial set of nodes is generated, capturing essential spatial entities (e.g., individual rooms). Additional nodes corresponding to corridor and outdoor areas are then introduced systematically to ensure a high-resolution coverage of navigable spaces.

Once the major structural regions have been established, a specialized door detection model identifies and classifies doors, which serve as critical connectors between rooms, corridors, and outdoor areas. The detected door locations are then used to anchor connectivity relationships by linking adjacent spatial regions, enabling the construction of an initial graph representation over the segmented floor plan. This preliminary graph captures the topological layout of the environment, including key transitions between enclosed and navigable spaces. To improve efficiency and usability, the graph is subsequently refined through shortest-path analysis and structural pruning, retaining only the minimal set of nodes and edges required to preserve full navigational connectivity.

#### 3.1.1 *Assumptions and Constraints.*

To ensure consistent and reliable performance, the *Tesseract* operates under a set of assumptions regarding the input floor plans. Annotations must be clear and explicit, with rooms, corridors, and outdoor areas distinctly labeled. Unlabeled or ambiguous regions are excluded or flagged for manual review. Room identifiers (e.g., "101", "102A") must be unique within each floor plan to prevent ambiguity during text extraction and graph generation. A room is defined as a bounded polygonal region enclosed by walls and containing at least one identifiable door. Open-concept areas without defined boundaries are excluded for consistency. Floor plans should represent only core structural elements such as walls, doors, and spatial boundaries, while omitting interior details like furniture, icons, or elevation markers. This assumption simplifies structural parsing and reduces false detections. Although not all floor plans strictly conform to these conditions, many sources such as evacuation maps, blueprints, and real estate diagrams are sufficiently aligned. These constraints ensure that each component of the system, from text recognition to graph construction, functions under well-defined and consistent conditions.

### 3.2 Floor Plan Patching and Preprocessing

The first stage of the *Tesseract* involves preprocessing large-scale floor plan images using the *Floor Plan Patcher* module. Raw floor plans—particularly those corresponding to multi-wing buildings or campus-scale facilities—often exceed 5000×5000 pixels in resolution, making direct processing computationally expensive and error-prone for vision-based models. Moreover, the fixed input size constraints of deep learning modules (e.g., CRAFT and Faster R-CNN) necessitate consistent image dimensions to maintain spatial context and recognition fidelity.

Therefore, each high-resolution input image is automatically partitioned into a grid of non-overlapping 1024×1024 pixels patches. This patching strategy ensures uniformity during model inference, improves GPU memory efficiency, and localizes the detection scope to regions with a manageable visual footprint. Additionally, it minimizes the likelihood of detection failures due to scale variation or contextual dilution—issues that frequently arise when processing very large spatial canvases in a single pass.

Each patch is processed independently through the downstream stages of the system (text detection, segmentation, door detection, etc.), and results are later reassembled into a unified graph representation. This modular decomposition preserves the spatial integrity of the original layout while enabling scalable, parallelized processing across varied architectural forms.

### 3.3 Text Detection and Spatial Segmentation

After patching high-resolution floor plans into standardized image segments, *Tesseract* focuses on two critical tasks: *text detection* and *floodfill-based spatial segmentation*. By accurately identifying textual labels (e.g., room identifiers) and delineating corresponding spatial regions, this step establishes the fundamental building blocks for graph construction and subsequent navigation.

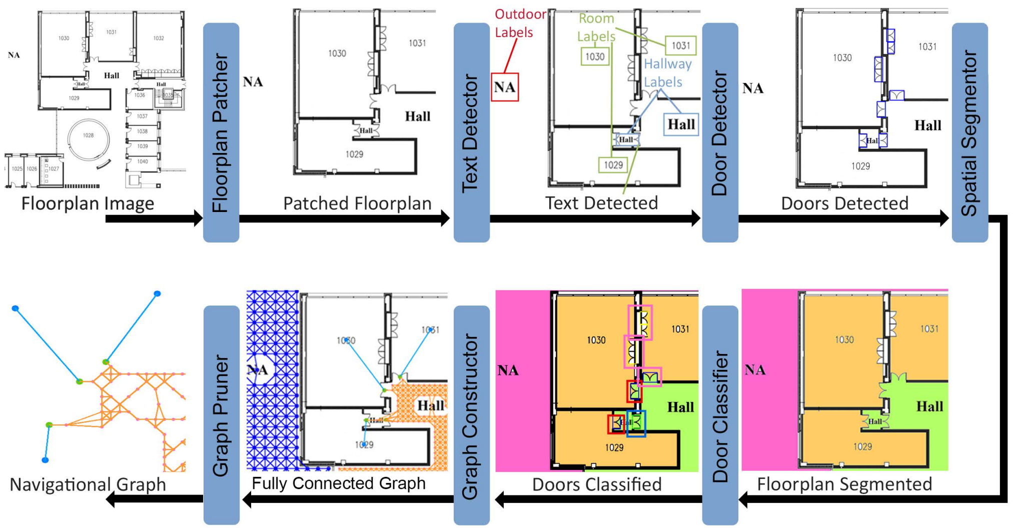

**Figure 1: Overview of the *Tesseract*, highlighting key modules: floor plan patching, text detection, door detection and classification, spatial segmentation, graph construction, and pruning to generate a final pruned graph representation.**

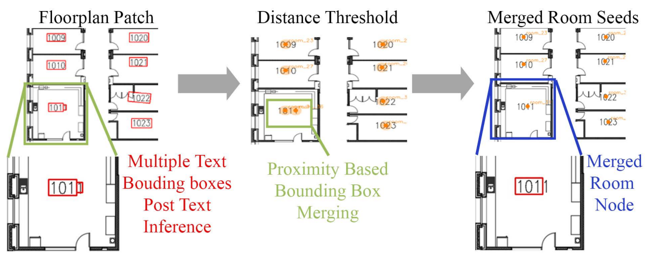
Floorplan Patch
Distance Threshold
Merged Room Seeds
Multiple Text Bouding boxes Post Text Inference
Proximity Based Bounding Box Merging
Merged Room Node

**Figure 2: Post-processing of text detection: multiple bounding boxes resulting from fragmented text inference (left) are merged based on spatial proximity (center), yielding a unified room node seed (right).**

*Text Detection.* We begin by applying the pre-trained CRAFT (Character Region Awareness for Text Detection) MLT-25K [3] model to localize textual elements within the floor plan. Specifically, three main categories of labels are extracted:

(1) *Room identifiers* (e.g., "1001", "102A"),
(2) *Corridor or hallway labels*, and
(3) *Ancillary annotations* such as exits or directional markers.

After generating bounding boxes for each detected text instance, a distance-based merging algorithm unifies fragmented boxes. For instance, if CRAFT partially segments a room number into two bounding boxes (e.g., "1011" being split into "101" and "1"), the algorithm identifies these bounding boxes as overlapping and merges them into a single detection (refer to Figure 2).

Refined bounding boxes are passed to a VGG BiLSTM-CTC model [2], trained on a domain-specific dictionary to extract textual content. Non-essential text (e.g., scale bars, markers) is discarded, while valid labels are retained. The *center* of each bounding box corresponding to a room, corridor, or outdoor label is used as a node candidate for graph construction.

*Floodfilling for Spatial Segmentation.* Once textual labels are identified, the system employs a seed-based floodfilling algorithm to segment the floor plan image into coherent, labeled regions. Each bounding box center acts as a primary seed, supplemented by additional seed points distributed radially (with radius $r = 20$ px and angular step size $\Delta\theta = 5px$ ). This radial seeding strategy helps overcome issues where the bounding box center might fall on the interior of a text label (e.g., inside the "loop" of a digit), preventing a proper floodfill initiation.

Floodfilling progresses outward from each seed until encountering black pixels that denote walls or other impassable barriers. This ensures that the resulting segmented areas remain true to the architectural boundaries. Once the floodfill completes, all pixels attributed to a particular seed are linked to the node corresponding to that label (e.g., a room node). Figure 3 illustrates the outcome of these segmentation steps in a representative floor plan.

Following segmentation, region boundaries and sizes are analyzed for consistency. Unassigned pixels, if any, are flagged for further review, ensuring comprehensive coverage across the image. These refined labels and segmented regions thus form a foundational layer for the subsequent graph-building process. In the next section, we discuss how these segmentation outputs are integrated with corridor and outdoor node placements, enabling a richer representation of the navigable spaces within the floor plan.

Algorithm 1 summarizes the text detection and seeded floodfilling procedure.

## 3.4 Graph Initialization and Door Detection

Building on the labeled and segmented regions established in Section 3.3, the next stage of the *Tesseract* constructs an initial graph representation and identifies doors that serve as critical connectors between spatial units. To capture the full navigable space, additional nodes are systematically placed within corridor and outdoor regions. Detected doors are then classified and used to define edges

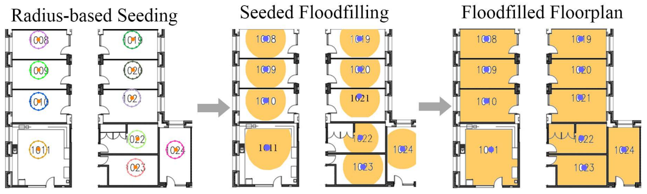

**Figure 3: Floodfilling process for spatial segmentation: initial seed points are distributed radially around each text label (left), region growing is performed via seeded floodfilling (center), and the resulting labeled regions form the segmented floor plan (right).**

between adjacent regions, enabling the construction of a coherent and well-connected topological graph.

*Populating Corridor and Outdoor Nodes.* After determining room labels and their corresponding floodfilled regions, a *wall mask* is generated from the floor plan image to identify structural boundaries. To prevent node placements from overlapping with walls or other impassable features, the wall mask is expanded (buffered) by a margin $\delta$. Only pixels outside this buffered region are deemed valid for node placement.

Within the valid area, corridor and outdoor nodes are positioned at regular intervals according to a grid-based stepping mechanism. Each grid cell of size $s \times s$ (with $s = 20\,\text{px}$) is sampled to yield uniformly spaced nodes, facilitating smooth navigation in subsequent graph operations. By tagging each node with its region type (*corridor* or *outdoor*), the algorithm differentiates navigable spaces from enclosed rooms, creating a cohesive substrate for connectivity analysis (see Figure 4).

*Door Detection and Classification.* Doors link rooms, corridors, and outdoor regions, making their detection and classification essential for reliable graph construction. We employ a fine-tuned Faster R-CNN model[34] to locate doors within the floor plan, leveraging

**Algorithm 1** Text Detection and Floodfilling with Pixel Assignment

```
1: Input: Floor plan image $I$
2: Output: Segmented regions with labeled pixels
3: procedure TextDetection($I$)
4:     Detect bounding boxes $\mathcal{B} = \{b_1, b_2, \dots, b_k\}$ using CRAFT MLT-25K.
5:     for all $(b_i, b_j) \in \mathcal{B}, i \neq j$ do
6:         if dist($b_i$.center, $b_j$.center) $< \epsilon$ then
7:             Merge $b_i$ and $b_j$ into one box.
8:         end if
9:     end for
10:    Refine $\mathcal{B}$, infer text $T(b)$ for each $b \in \mathcal{B}$ using VGG BiLSTM-CTC.
11:    Retain $b \in \mathcal{B}$ where $T(b) \in \{\text{Room, Hallway, Outdoor}\}$.
12:    Extract nodes $\mathcal{N} = \{b.\text{center} \mid b \in \mathcal{B}\}$.
13: end procedure
14: procedure Floodfilling($\mathcal{N}, I$)
15:    for all $n \in \mathcal{N}$ do
16:        Create circle $C(n, r)$ of radius $r$ around $n$.
17:        Generate seeds $\mathcal{S}_n = \{s_\theta \mid s_\theta = n + r(\cos\theta, \sin\theta), \theta \in [0, 2\pi), \Delta\theta\}$.
18:        for all $s \in \mathcal{S}_n$ do
19:            Floodfill from $s$, stopping at black pixels.
20:            Assign floodfilled pixels to node $n$.
21:        end for
22:    end for
23:    Compute region boundaries and sizes from floodfilled areas.
24:    Flag unclassified regions for further inspection.
25: end procedure
26: Return: Segmented regions with labeled pixels
```

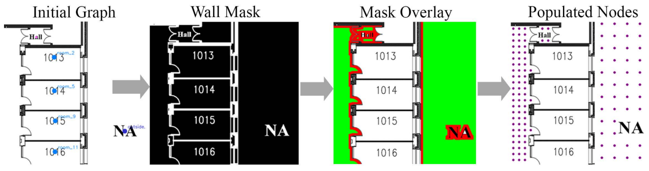

**Figure 4: Graph node population process: initial room nodes are extracted (left), walls are masked and buffered (center), and valid corridor/outdoor regions are populated with uniformly spaced nodes (right).**

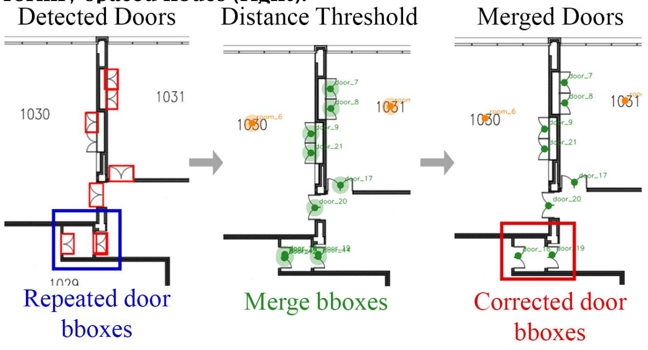

**Figure 5: Post-processing of door detection: repeated bounding boxes (left) are merged using a proximity threshold (center), resulting in corrected door annotations (right).**

its region proposal network to generate initial bounding boxes, accompanied by confidence scores. A post-processing routine then refines and merges overlapping predictions based on a defined proximity threshold, ensuring that door detections align with structural features (see Figure 5).

Each finalized bounding box is classified using the region labels obtained from floodfilling (Section 3.3), allowing the system to identify the door's connectivity role within the layout. Bounding boxes that do not yield valid spatial transitions are discarded.

Each valid door then introduces an edge between adjacent spatial entities (e.g., *room–door* or *corridor–door*), contributing to the initial connected graph. In the next section, we describe how these edges and nodes are refined through graph-level optimization.

Algorithm 2 summarizes the door detection and classification procedure.

## 3.5 Graph Construction and Optimization

Having assigned nodes to each region and classified the doors that link these nodes, the next step in the *Tesseract* is to form a cohesive graph representation of the floor plan. This graph serves as the underpinning for navigable indoor maps and spatial queries, capturing both localized connectivity (e.g., room-to-door) and broader spatial relationships (e.g., corridor traversal). To maintain clarity and computational efficiency, the graph undergoes an optimization phase that prunes redundant edges while preserving critical paths.

*Edge Creation.* Edges are constructed to capture the following essential spatial relationships:

- **Room-to-Door Connectivity:** Each room node is linked to all doors that share its boundary, as determined by pixel-level adjacency from the floodfilling stage.

- **Door-to-Hallway Connectivity:** Doors connecting a room to a corridor are used to create edges between the door node and the relevant corridor node.
- **Hallway Connectivity:** Corridor nodes are interconnected based on spatial proximity, ensuring continuous navigation throughout the hallway network.
- **Exit Connections:** Exits are joined to outdoor nodes to facilitate direct transitions between indoor and outdoor areas.

Each edge is annotated with metadata (e.g., *Room-to-Door*, *Corridor-to-Corridor*) to support targeted queries and pathfinding tasks.

*Graph Optimization via Shortest Paths and Structural Pruning.* Although the initial graph encompasses all necessary nodes and edges, some connections may be redundant, complicating navigation and increasing computational overhead. To address this, we employ a hybrid optimization approach that combines shortest-path calculations with strategic pruning:

(1) *Initial Graph Construction:* All regions (rooms, corridors, and outdoor areas) are linked through their detected doors, producing a fully connected structure.
(2) *Shortest Paths Computation:* The algorithm calculates the shortest path between every pair of rooms, retaining the minimal set of edges and nodes that sustain connectivity.
(3) *Exit Connectivity:* Each room is also linked to its nearest exit door via the shortest path, ensuring compliance with evacuation and accessibility requirements.
(4) *Pruning Non-Essential Edges:* Any connection not involved in maintaining room-to-room or room-to-exit accessibility is removed, leaving a streamlined graph that accurately reflects core spatial relationships.

By preserving only the paths essential for connectivity, the resulting graph strikes a balance between navigational accuracy and computational efficiency. Figure 6 illustrates an example of such a pruned graph, showing how redundant connections are eliminated to simplify the underlying structure. The overall graph construction and pruning procedure is summarized in Algorithm 3.

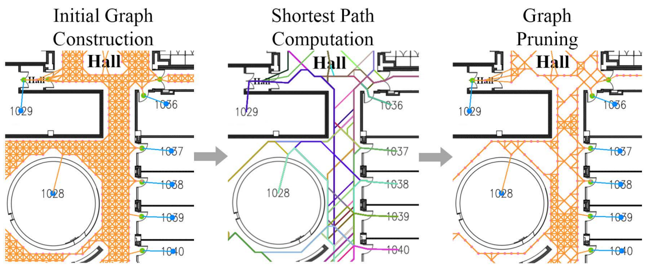

**Figure 6: Graph pruning: the initial fully connected graph is refined to remove redundant edges, yielding a streamlined, navigable representation.**

**Algorithm 2** Door Detection and Classification

```
1:  Input: Floor plan image $I$, Regions $\mathcal{R}$, Graph nodes $\mathcal{N}$
2:  Output: Updated graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ with classified doors
3:  procedure DoorDetectionAndClassification($I, \mathcal{N}, \mathcal{R}$)
4:      Detect doors $\mathcal{D} = \{d_1, d_2, \ldots, d_m\}$ using a deep learning model.
5:      for all door $d \in \mathcal{D}$ do
6:          Refine $d$.bbox to align with $I$.
7:          Extract pixel regions $P_{\text{left}}, P_{\text{right}}$ on opposite sides of $d$.bbox.
8:          Determine region types:
    $\text{type}_{\text{left}} = \text{RegionType}(P_{\text{left}}, \mathcal{R})$, $\text{type}_{\text{right}} = \text{RegionType}(P_{\text{right}}, \mathcal{R})$.
9:          if $\text{type}_{\text{left}} = \text{Outdoor} \land \text{type}_{\text{right}} \in \{\text{Room}, \text{Corridor}\}$ then
10:             Set $d$.type = Exit.
11:         else if $\text{type}_{\text{left}} = \text{Corridor} \land \text{type}_{\text{right}} = \text{Corridor}$ then
12:             Set $d$.type = Corridor-to-Corridor.
13:         else if $\text{type}_{\text{left}} = \text{Room} \land \text{type}_{\text{right}} = \text{Room}$ then
14:             Set $d$.type = Room-to-Room.
15:         else if $\text{type}_{\text{left}} = \text{Room} \land \text{type}_{\text{right}} = \text{Corridor}$ then
16:             Set $d$.type = Room-to-Corridor.
17:         else
18:             Discard $d$.
19:         end if
20:     end for
21:     Add edges $(n_1, n_2) \in \mathcal{E}$ for all valid $d \in \mathcal{D}$, connecting nodes $n_1, n_2$ adjacent
    to $d$.bbox.
22: end procedure
23: Return: Updated graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$
```

**Algorithm 3** Graph Construction and Optimization

```
1:  Input: Nodes $\mathcal{N}$, Doors $\mathcal{D}$, Adjacency $\mathcal{A}$
2:  Output: Optimized graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$
3:  procedure GraphConstruction($\mathcal{N}, \mathcal{D}, \mathcal{A}$)
4:      Initialize edges $\mathcal{E} = \emptyset$
5:      for all room $r \in \mathcal{N}$ do
6:          for all door $d \in \mathcal{D}$ do
7:              if $d$ adjacent to $r$ then
8:                  Add edge $(r, d)$ to $\mathcal{E}$              ▷ Room-to-Door connection
9:              end if
10:         end for
11:     end for
12:     for all door $d_1, d_2 \in \mathcal{D}$ do
13:         if type$(d_1, d_2)$ = Corridor then
14:             Add edge $(d_1, d_2)$ to $\mathcal{E}$               ▷ Hallway connection
15:         end if
16:     end for
17:     for all exit door $e \in \mathcal{D}$ do
18:         Add edge $(e, r)$ for the nearest room $r \in \mathcal{N}$
19:     end for
20: end procedure
21: procedure GraphOptimization($\mathcal{G}$)
22:     Initialize $\mathcal{E}_{\text{opt}} = \emptyset$
23:     for all pair of rooms $(r_1, r_2) \in \mathcal{N}$ do
24:         Compute shortest path $\mathcal{P}(r_1, r_2)$ in $\mathcal{G}$
25:         Add nodes and edges in $\mathcal{P}(r_1, r_2)$ to $\mathcal{E}_{\text{opt}}$
26:     end for
27:     for all room $r \in \mathcal{N}$ do
28:         Connect $r$ to the nearest exit $e \in \mathcal{D}$ via shortest path
29:         Add corresponding edges to $\mathcal{E}_{\text{opt}}$
30:     end for
31:     Prune edges not in $\mathcal{E}_{\text{opt}}$ from $\mathcal{G}$
32: end procedure
33: Return: Optimized graph $\mathcal{G} = (\mathcal{N}, \mathcal{E}_{\text{opt}})$
```

This optimized representation forms the final output of the *Tesseract*, supporting various applications including indoor route planning, area surveillance, and space utilization analysis. In the following section, we describe the experimentation framework used to evaluate the proposed approach, detailing the datasets, setup, and performance metrics employed in our empirical studies.

## 4 Evaluation

### 4.1 Input Data and Setup

To evaluate the effectiveness and scalability of our proposed *Tesseract*, we conduct experiments on three distinct floor plan datasets. Each dataset varies in complexity, size, and annotation density, allowing us to assess performance across diverse architectural layouts.

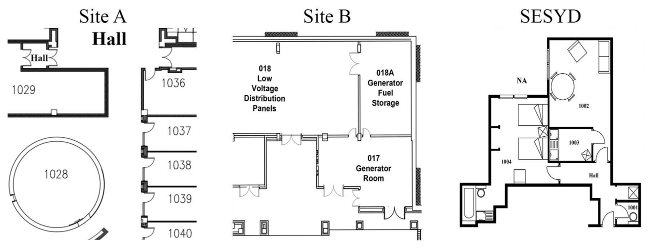

**Figure 7: Floor plan samples from the three evaluation datasets: Site A (left), Site B (center), and SESYD (right).**

All datasets undergo a uniform preprocessing routine involving resolution normalization, noise reduction, and contrast enhancement to improve text detection and segmentation accuracy.

*Datasets.* We draw on two large-scale floor plan datasets, referred to as *Site A* and *Site B*, as well as a smaller-scale dataset from *SESYD*, summarized in Table 1 and visualized in Figure 7.

- **Site A:** Comprising three high-resolution floor plan images, each covering an extensive indoor area with more than 15 annotated rooms or regions. These images are subdivided into a total of eight patches with varying dimensions, reflecting diverse structural layouts and wall arrangements.
- **Site B:** Consisting of four large-scale floor plans sampled at a fixed patch size, yielding four patches in total. This consistent segmentation enables a uniform approach to feature extraction and allows fair comparisons of algorithmic performance across similarly sized patches.
- **SESYD:** A collection of ten smaller, more structured architectural floor plans. Each image is equally sized and retains a fixed patch dimension, providing a benchmark for testing the pipeline on compact and systematically annotated spaces.

*Implementation Details.* All experiments are carried out on a Linux-based high-performance system equipped with two NVIDIA A6000 GPUs, an Intel Core i7 CPU, and 128 GB of RAM.

The *Tesseract* processes each floor plan (or patch) independently. Text detection and recognition modules run on the GPU-accelerated frameworks, followed by CPU-based routines for wall masking, node generation, and graph construction. Typical runtime per floor plan patch ranges from 1–3 minutes, depending on image resolution and the complexity of the floor plan topology.

## 4.2 Metrics and Guidelines for Results

The proposed *Tesseract* is evaluated using metrics of *navigability*, *efficiency*, and *graph quality*, capturing its ability to produce accurate, compact, and traversable indoor floor plan graphs.

*4.2.1 Navigability and Graph Completeness.* Producing a graph representation that reflects the floor plan's structural connectivity is critical. To quantify this, we define two complementary metrics: *navigational completeness* and *exit reachability*.

*Navigational Completeness.* Let $\mathcal{R}$ be the set of all room nodes in the graph. For any pair of distinct rooms $(r_i, r_j) \in \mathcal{R}$, we define reachable$(r_i, r_j)$ as a boolean function that evaluates to true if a

**Table 1: Summary of Floor Plan Datasets**

| Attribute | Site A | Site B | SESYD |
|---|---|---|---|
| # Floor Plans | 3 | 4 | 10 |
| # Patches | 22 | 18 | 10 |
| Patch Size (px) | Variable | 1500x1500 | 1024x1024 |
| Resolution (px/m) | 300 | 200 | 150 |
| Avg. Rooms/Patch | 124 | 87 | 8 |

valid path exists between $r_i$ and $r_j$ in the graph. Navigational completeness, denoted by $\eta \in [0, 1]$, is computed as the proportion of mutually reachable room pairs:

$$\eta = \frac{\sum_{(r_i, r_j) \in \mathcal{R}} \mathbf{1}(\text{reachable}(r_i, r_j))}{|\mathcal{R}| \times (|\mathcal{R}| - 1)},$$

where $\mathbf{1}(\cdot)$ is the indicator function. This metric is further extended to verify *exit accessibility* by ensuring that each room node is also connected to at least one designated exit node.

A curated ground-truth graph, derived from manual inspection of each floor plan, serves as the benchmark for completeness. Comparing the generated graph to this ground truth helps confirm that no rooms or subgraphs remain isolated and that the final output aligns with the expected architectural layout.

*Exit Reachability.* In addition to inter-room connectivity, we also evaluate whether each room has access to at least one exit. Let $\mathcal{E}$ denote the set of exit door nodes. For every room node $r_i \in \mathcal{R}$, we define reachable$(r_i, \mathcal{E})$ as true if a valid path exists from $r_i$ to any node in $\mathcal{E}$. Exit reachability is then measured as the proportion of rooms from which at least one exit is reachable:

$$\epsilon = \frac{\sum_{r_i \in \mathcal{R}} \mathbf{1}(\text{reachable}(r_i, \mathcal{E}))}{|\mathcal{R}|}.$$

This metric provides a practical check on the usability of the graph for safety-critical tasks such as evacuation planning.

*4.2.2 Computational Efficiency.* We evaluate computational performance by measuring the total runtime per floor plan or patch, from raw input to final graph. This end-to-end runtime encompasses:

- **Preprocessing Overhead:** Image binarization and text annotation stages.
- **Segmentation and Node Creation:** Floodfilling, label filtering, and node/seed generation.
- **Graph Construction:** Door detection, node-to-node linking, and metadata assignment.
- **Optimization:** Shortest-path calculations, pruning strategies, and final data serialization.

By reporting both the overall and per-stage runtimes, we identify potential bottlenecks (e.g., door detection on large, complex floor plans) and demonstrate the scalability of the *Tesseract* across datasets with varying sizes and complexities.

*4.2.3 Graph Structure and Optimization Impact.* To quantify how effectively the system balances representational detail with minimal redundancy, we track fundamental graph characteristics:

- **Node and Edge Counts:** The total number of nodes ($|\mathcal{N}|$) and edges ($|\mathcal{E}|$) provides an immediate measure of complexity. A high node or edge count may indicate over-segmentation or overly detailed corridor sampling, whereas excessively low counts risk compromising navigability.

- **Pruning Effectiveness:** We compare the node and edge counts before and after applying shortest-path-based pruning. A significant reduction without a drop in navigational completeness suggests that unnecessary pathways have been successfully removed.
- **Storage Footprint:** Since the final graph is exported in a JSON-based format, the file size offers a straightforward gauge of representational compactness. Marked decreases in JSON size indicate that the system has condensed the floor plan structure into an efficient yet navigable graph.

*4.2.4 Geometric Fidelity.* When travel times or route distances matter for applications such as evacuation planning, *geometric fidelity* becomes a critical factor. We assess how well the final graph's path distances align with the floor plan's spatial reality:

- **Pixel-Based Ground Truth:** For each pair of room centers $(r_i, r_j)$, we measure the Euclidean distance $\text{dist}_{\text{floor plan}}(r_i, r_j)$ directly from the annotated floor plan image.
- **Graph-Based Distance:** We then compute the shortest path $\text{dist}_{\text{graph}}(r_i, r_j)$ in the final graph, accounting for corridor traversal and door transitions.
- **Distance Ratio:** We use the ratio $\rho(r_i, r_j) = \frac{\text{dist}_{\text{graph}}(r_i, r_j)}{\text{dist}_{\text{floor plan}}(r_i, r_j)}$ to quantify how closely the graph-based path length approximates the actual floor plan distance. A ratio near 1.0 indicates minimal distortion, while larger deviations suggest either oversimplification or unmodeled detours.

Aggregating these ratios over all room pairs in $\mathcal{R}$ yields a mean distance ratio $\bar{\rho}$ that provides a measure of geometric fidelity.

By integrating these metrics, our evaluation framework captures both the *accuracy* of the resulting indoor graph—reflected in navigability, completeness, and geometric fidelity—and its *efficiency*—demonstrated through runtime, graph compactness, and minimized redundancy. In the subsequent section, we present qualitative and quantitative results for each of the three datasets introduced in Section 4.1, examining how *Tesseract* performs under diverse architectural scales and configurations.

## 5 Results

### 5.1 Navigability and Graph Completeness

This subsection presents an in-depth analysis of how effectively the proposed *Tesseract* preserves connectivity across rooms and exits in the final, pruned graph. We first report the overall *navigational completeness* for each dataset—*Site A*, *Site B*, and *SESYD*—and then delve into *exit reachability* $\epsilon$, examining the extent to which all rooms can access designated exit doors.

*Overall Connectivity.* Table 2 provides a summary of navigational completeness scores, defined as the proportion of pairwise room connections present in the final graph. These values reflect how well *Tesseract* captures inter-room paths and corridor transitions given its text-detection, floodfilling, and door-classification modules.

*SESYD* achieves the highest navigational completeness, nearing a perfect score of 0.98 (Table 2). This strong connectivity stems from the dataset's simpler and more structured layouts, which reduce

**Table 2: Navigational Completeness and Exit Reachability**

| Dataset | Navigational Completeness | Exit Reachability | Main Sources of Disconnection |
|---|---|---|---|
| **Site A** | 0.93 ± 0.02 | 0.90 ± 0.03 | Missed Doors, Partial Floodfill |
| **Site B** | 0.90 ± 0.03 | 0.86 ± 0.04 | High Door Density, Patch Splits |
| **SESYD** | 0.98 ± 0.01 | 0.96 ± 0.01 | Occasional Door Misdetections |

the likelihood of ambiguous boundaries or overlapping door placements. The few disconnections observed (roughly 2%) primarily arise from missed doors, typically when the Faster R-CNN model fails to detect small or partially occluded doorways.

*Site A* and *Site B* both exhibit slightly lower completeness scores, attributed to their larger, more complex floor plans, and higher variability in room sizes. In *Site A*, the average completeness of 0.93 reflects occasional failures in door detection along extremely long corridors. In some cases, overly conservative floodfilling around thick walls contributes to incomplete room connectivity. Meanwhile, *Site B* shows a marginally lower mean completeness (0.90), which appears tied to its smaller patch sizes and the high density of doors per patch. When a door is missed in such a confined patch, entire room clusters risk isolation unless alternative routes exist.

*Exit Reachability.* Exit reachability, also listed in Table 2, evaluates whether every room node in the final graph can access a designated exit node. For *SESYD*, 96% of rooms achieve exit connectivity, reaffirming the dataset's relatively simple layouts. By contrast, *Site A* and *Site B* demonstrate a modest drop in exit reachability (0.90 and 0.86, respectively) compared to their overall completeness. This discrepancy highlights that while many rooms remain interconnected, certain missed or misclassified doors near external boundaries can sever paths leading outside. In *Site B*, doors that appear only within patch overlaps occasionally go undetected, impeding the connection between entire interior segments and exit nodes.

Overall, these results confirm that our approach yields high navigational completeness in simpler architectural settings and remains robust in larger, more complex sites. Nevertheless, improving door detection accuracy—particularly for edge cases where doors are partially obscured or conflated with text—could further enhance connectivity. In the next subsection, we discuss computational efficiency results and explore how the pipeline's runtime scales with floor plan size and patch density.

### 5.2 Computational Efficiency

The runtime performance of the *Tesseract* is examined by breaking down execution times into four modules: *(i) Preprocessing and Segmentation*, *(ii) Door Detection*, *(iii) Door Classification*, and *(iv) Graph Construction and Optimization*. Table 3 summarizes the average runtimes for each module across our three datasets, along with the total end-to-end processing time. As shown in Figure 8, processing time approximately scales linearly with the number of detected text labels, with large-scale datasets exhibiting higher runtimes due to increased scene complexity.

*Overall Runtime Trends.* As shown in Table 3, door detection and door classification consistently consume the largest portion

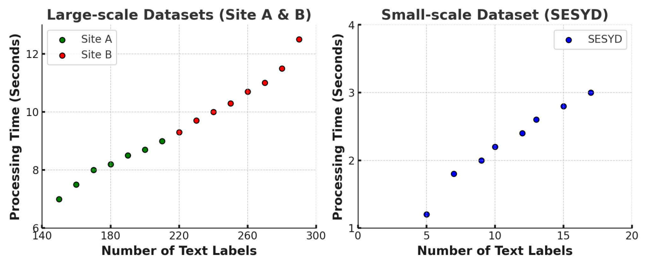

**Figure 8: Processing time vs. number of text labels. Sites A & Site B exhibit linear scaling with floor plan complexity, while SESYD (right) shows lower runtime due to simpler layouts.**

**Table 3: Per-Module and Total Average Runtime (in seconds). Values represent the mean ± standard deviation across the patches in each dataset.**

| Module | Site A | Site B | SESYD |
|---|---|---|---|
| Preprocessing Segmentation | 9.8 ± 1.7 | 12.1 ± 2.3 | 6.2 ± 1.3 |
| Door Detection | 36.2 ± 3.8 | 40.5 ± 4.7 | 30.7 ± 2.6 |
| Door Classification | 22.6 ± 2.6 | 25.1 ± 2.9 | 16.4 ± 2.2 |
| Graph Const. Optimization | 13.7 ± 2.0 | 15.7 ± 3.1 | 10.2 ± 1.3 |
| **Total** | **82.3 ± 10.1** | **93.4 ± 9.9** | **63.5 ± 5.8** |

**Table 4: Impact of Contrast Enhancement on Text Merging. Values represent the average number of bounding-box merges needed per patch *before* and *after* preprocessing.**

| Dataset | Preprocessing Off | Preprocessing On | Reduction % |
|---|---|---|---|
| Site A | 27.2 ± 3.5 | 4.1 ± 1.1 | 85% |
| Site B | 19.8 ± 2.8 | 3.6 ± 1.0 | 82% |
| SESYD | 12.3 ± 1.7 | 1.9 ± 0.6 | 85% |

of processing time, especially in *Site B*, which features numerous smaller regions and a high density of doorways. Here, detecting and subsequently classifying every door can account for more than half of the total runtime. By contrast, *SESYD* benefits from fewer rooms per patch, reducing the time spent on both bounding-box generation and classification.

*Preprocessing and Segmentation.* While preprocessing (resolution normalization and contrast enhancement) costs relatively little time, it substantially reduces fragmented text detections in subsequent steps. Table 4 illustrates how increasing contrast lowers the average number of text bounding-box merges by over 80% across all datasets, thereby accelerating downstream text recognition.

For segmentation, floodfilling remains efficient even in *Site A* and *Site B*. Most segmentation issues stem from structural complexities like auditorium seating, rather than wall thickness per se. These obstacles can occasionally lead to partial floodfills, with potential ramifications for connectivity if not rectified later in the system.

*Door Detection and Classification.* In larger-scale floor plans, door detection occupies between 30 to 40 seconds on average, reflecting the time spent identifying bounding boxes for a high volume of potential door candidates. The subsequent door classification step entails examining pixel-level context (room vs. corridor vs. outdoor) for each bounding box, and hence dominates overall runtime in datasets with many closely spaced doors. For instance, *Site B* exhibits particularly prolonged classification times (25.1±2.9 seconds), as multiple small patches with dense door placements necessitate repeated context checks.

*Graph Construction and Optimization.* Constructing the graph by linking room, corridor, door, and exit nodes is relatively efficient, averaging about 10–15 seconds in total across all datasets. Once complete, graph optimization prunes redundant edges and calculates shortest paths to solidify connectivity. Though this final phase is shorter in duration (roughly 10–20% of total runtime), it yields important benefits: consolidated node sets and streamlined edge topologies facilitate more efficient routing queries during subsequent use cases.

Overall, the *Tesseract* remains computationally viable for both large-scale and smaller datasets, maintaining an approximate end-to-end runtime under two minutes per patch even in the most complex scenarios. The next subsection (Section 5.3) discusses how these constructed and pruned graphs manifest in practice, highlighting node/edge distributions and the corresponding storage benefits of removing extraneous connections.

**Table 5: Node and Edge Counts Before and After Pruning (Mean ± SD). Reduction % is the proportion of nodes/edges removed relative to the original graph.**

| Data | Nodes | | Edges | | Node Red % | Edge Red % |
|---|---|---|---|---|---|---|
| | Before | After | Before | After | | |
| **Site A** | 2088 ± 280 | 459 ± 72 | 5820 ± 344 | 1512 ± 98 | 78% | 74% |
| **Site B** | 1221 ± 190 | 420 ± 58 | 4230 ± 261 | 1144 ± 85 | 66% | 73% |
| **SESYD** | 85 ± 16 | 52 ± 10 | 210 ± 30 | 136 ± 18 | 39% | 35% |

## 5.3 Graph Structure and Pruning Effectiveness

As noted in Section 5.2, graph construction and optimization account for only a small share of runtime but are essential for producing compact, navigable indoor graphs. This subsection highlights the achieved *node and edge* reductions from pruning and the corresponding *storage savings* in the final JSON output.

*Node and Edge Counts Before & After Pruning.* Following door classification, each patch is initially represented by a fully connected graph encompassing all discovered rooms, corridors, outdoor nodes, and doors. Table 5 illustrates how optimization removes redundant edges and, in some cases, merges nodes representing near-identical positions in the corridor or hallways. As a result, the final graph experiences a considerable reduction in overall complexity without impairing navigational completeness (see Section 5.1).

Notably, *Site A* and *Site B* exhibit more substantial edge reductions (over 74%) and node reduction (over 78%), largely due to overlapping corridor nodes and door connections that can be pruned without impacting reachability. Although *SESYD* benefits less dramatically from pruning (around 35% of edges removed and around 39% of nodes removed), it still sees modest simplification gains owing to its smaller scale and lower corridor redundancy. Crucially, none of these modifications degrade navigational completeness:

**Table 6: Average JSON File Size Before and After Pruning.**

| Dataset | Before Pruning (KB) | After Pruning (KB) | Reduction % |
|---|---|---|---|
| Site A | 889 ± 85 | 218 ± 28 | 75% |
| Site B | 905 ± 79 | 225 ± 25 | 71% |
| SESYD | 345 ± 40 | 45 ± 11 | 87% |

the system ensures that every valid route from a room to another (or to an exit) persists post-optimization.

*Final Graph Storage Footprint.* To quantify how these structural reductions translate into file size savings, we store each patch's graph in a JSON-based format and measure its size before and after pruning. Table 6 shows that all datasets see meaningful file-size reductions, reflecting the removal of extraneous edges and consolidated node representations.

In *Site A*, broader corridors and dense door connectivity result in an average file-size reduction of nearly 75%. For *Site B*, finer patch segmentation leads to the removal of many corridor-to-door edges, reducing storage by 71%. In *SESYD*, with fewer true room nodes and edges, the final graph is on average 87% smaller.

Taken together, these results underscore the effectiveness of pruning for eliminating needless complexity in corridor-dense or door-rich environments, facilitating both *storage efficiency* and *ease of navigation*. Notably, the *Tesseract*'s design ensures that no critical route or exit pathway is removed during optimization, preserving the high connectivity reported in Section 5.1. Consequently, the final graphs remain well-suited for tasks like evacuation routing and indoor wayfinding, while maintaining compact file representations that can be rapidly loaded or transmitted. In the next section, we analyze selected floor plans visually and discuss the qualitative aspects of the pruned graphs.

## 5.4 Geometric Fidelity

Although the focus of our evaluation primarily centers on navigability and structural completeness, certain applications (e.g., detailed evacuation planning or resource allocation) may demand that the final graph accurately reflects *real-world distances*. In such cases, we measure geometric fidelity by comparing the Euclidean distances directly derived from the floor plan with those computed along the shortest paths in the pruned graph.

Table 7 summarizes the mean and standard deviation of distance ratios for each dataset. These values are derived from at least 12-20 representative room pairs per floor plan in *Site A*, *Site B*, and *SESYD*.

Overall, *SESYD* demonstrates the smallest deviation from unity, with a mean ratio of 0.96 and a maximum deviation of 0.13. This near-ideal performance is attributable to the dataset's simpler layouts and smaller scale, where corridors are typically straight, and doors closely align with the conceptual "centers" of edges.

Conversely, *Site B* shows greater variance, with a maximum deviation of 0.28. This is primarily due to corridor sampling and door classification in areas with complex hallways or irregular rooms. In such cases, the shortest path within the graph sometimes marginally overestimates or underestimates true Euclidean distances, reflecting a partial "shortcut" effect or unavoidable detour.

Despite these deviations, most room pairs in both large-scale (*Site A*, *Site B*) and smaller-scale (*SESYD*) floor plans exhibit distance ratios above 0.90. For navigation and planning purposes, this level of geometric fidelity is typically sufficient, particularly if the primary objective is to ensure connectivity rather than precise distance measurement. Where more accurate distance estimates are critical (e.g., evacuation route timing), additional refinements—such as higher-resolution corridor sampling or post-processing adjustments to edge weights—can improve alignment between real-world and graph-based distances.

**Table 7: Average Distance Ratios and Their Standard Deviations Across Datasets. Ratios close to 1.0 reflect minimal distortion between floor plan and graph-based distances.**

| Dataset | Mean $\rho$ | Std. Dev. | Max. Deviation |
|---|---|---|---|
| **Site A** | 0.94 | 0.07 | 0.22 |
| **Site B** | 0.92 | 0.10 | 0.28 |
| **SESYD** | 0.96 | 0.04 | 0.13 |

Thus, while geometric fidelity may be optional in many practical settings, the results demonstrate that the *Tesseract* preserves distances reasonably well, especially in less convoluted architectural scenarios. Future work could incorporate room shape priors or multi-seed sampling tailored to hallway geometry to further narrow the gap between graph-based and true Euclidean distances.

## 5.5 Discussion

Quantitative results demonstrate that *Tesseract* consistently achieves high navigational completeness and substantial graph simplification across diverse architectural layouts. In both large-scale university buildings and compact, structured environments, the system maintains over 90% inter-room connectivity and reduces redundant graph elements by up to 78% in nodes and 74% in edges. These results validate its scalability and adaptability, showing that it generalizes well to varying levels of complexity without compromising performance.

Door detection and classification are the most critical factors in maintaining graph connectivity. Missed or misclassified doors, especially in dense areas with frequent transitions, can create isolated subgraphs or limit exit access. Structural complexities such as irregular walls, embedded fixtures, or narrow junctions may also reduce segmentation accuracy, causing incomplete boundaries or adjacency errors. Enhancing these modules with context-aware detection or geometric reasoning could improve robustness. Overall, however, the system remains highly reliable, offering a practical solution for large-scale indoor map generation and navigation.

## 6 Conclusion

This paper introduced *Tesseract*, a novel deployable system that constructs navigable indoor graphs solely from low-semantic static floor plan images. *Tesseract* circumvents the need for advanced sensors or specialized modeling software. By combining deep learning techniques for text and door detection with our intelligent floodfill-based floor plan segmentation, our system extracts semantically meaningful nodes and edges, culminating in a graph representation optimized through shortest-path pruning.

## References

[1] Raghad Alqobali, Maha Alshmrani, Reem Alnasser, Asrar Rashidi, Tareq Alhmiedat, and Osama Moh'd Alia. 2023. A survey on robot semantic navigation systems for indoor environments. *Applied Sciences* 14, 1 (2023), 89.

[2] Jeonghun Baek, Geewook Kim, Junyeop Lee, Sungrae Park, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. 2019. What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 4715–4723.

[3] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. 2019. Character Region Awareness for Text Detection. In *IEEE CVPR*. 9365–9374.

[4] Emanuele Bastianelli, Domenico Daniele Bloisi, Roberto Capobianco, Fabrizio Cossu, Guglielmo Gemignani, Luca Iocchi, and Daniele Nardi. 2013. On-line semantic mapping. In *IEEE ICAR*. IEEE.

[5] Hriday Bavle, Jose Luis Sanchez-Lopez, Claudio Cimarelli, Ali Tourani, and Holger Voos. 2023. From slam to situational awareness: Challenges and survey. *Sensors* 23, 10 (2023), 4849.

[6] Roberto Bigazzi, Lorenzo Baraldi, Shreyas Kousik, Rita Cucchiara, Marco Pavone, et al. 2024. Mapping High-level Semantic Regions in Indoor Environments without Object Recognition. In *Proceedings of the 2024 IEEE ICRA*.

[7] Jorge Chen and Keith C Clarke. 2020. Indoor cartography. *Cartography and Geographic Information Science* 47, 2 (2020), 95–109.

[8] Qi Chen, Qi Wu, Rui Tang, Yuhan Wang, Shuai Wang, and Mingkui Tan. 2020. Intelligent home 3d: Automatic 3d-house design from linguistic descriptions only. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12625–12634.

[9] Zhengyi Chen, Hao Wang, Keyu Chen, Changhao Song, Xiao Zhang, Boyu Wang, and Jack CP Cheng. 2024. Improved coverage path planning for indoor robots based on BIM and robotic configurations. *Automation in Construction* 158 (2024), 105160.

[10] David DeFazio, Hrudayangam Mehta, Jeremy Blackburn, and Shiqi Zhang. 2024. Vision Language Models Can Parse Floor Plan Maps. *arXiv preprint arXiv:2409.12842* (2024).

[11] Mathieu Delalandre, Ernest Valveny, Tony Pridmore, and Dimosthenis Karatzas. 2010. Generation of synthetic documents for performance evaluation of symbol recognition & spotting systems. *International Journal on Document Analysis and Recognition (IJDAR)* 13, 3 (2010), 187–207.

[12] Hugh Durrant-Whyte and Tim Bailey. 2006. Simultaneous localization and mapping: part I. *IEEE robotics & automation magazine* 13 (2006).

[13] Amr Eldemiry, Yajing Zou, Yaxin Li, Chih-Yung Wen, and Wu Chen. 2022. Autonomous exploration of unknown indoor environments for high-quality mapping using feature-based RGB-D SLAM. *Sensors* 22, 14 (2022), 5117.

[14] Aliaa Essameldin, Mohammed Nurul Hoque, and Khaled A Harras. 2020. More than the sum of its things: Resource sharing across iots at the edge. In *Proc. IEEE/ACM Symp. Edge Comput.* 206–219.

[15] Jonathan Francis, Nariaki Kitamura, Felix Labelle, Xiaopeng Lu, Ingrid Navarro, and Jean Oh. 2022. Core challenges in embodied vision-language planning. *Journal of Artificial Intelligence Research* 74 (2022), 459–515.

[16] Héctor H González-Banos and Jean-Claude Latombe. 2002. Navigation strategies for exploring indoor environments. *The International Journal of Robotics Research* 21, 10-11 (2002), 829–848.

[17] Omar Hashem, Khaled A Harras, and Moustafa Youssef. 2020. Deepnar: Robust time-based sub-meter indoor localization using deep learning. In *IEEE SECON*. IEEE, 1–9.

[18] Omar Hashem, Khaled A Harras, and Moustafa Youssef. 2021. Accurate indoor positioning using IEEE 802.11 mc round trip time. *Pervasive and Mobile Computing* 75 (2021), 101416.

[19] Omar Hashem, Moustafa Youssef, and Khaled A Harras. 2020. WiNar: RTT-based sub-meter indoor localization using commercial devices. In *2020 IEEE international conference on pervasive computing and communications (PerCom)*. IEEE, 1–10.

[20] Nathan Hughes, Yun Chang, and Luca Carlone. 2022. Hydra: A real-time spatial perception system for 3D scene graph construction and optimization. *Robotics: Science and Systems* (2022).

[21] Amir Ibrahim, Ali Sabet, and Mani Golparvar-Fard. 2019. BIM-driven mission planning and navigation for automatic indoor construction progress detection using robotic ground platform. In *EC3 Conference 2019*, Vol. 1. European Council on Computing in Construction, 182–189.

[22] Ismail Ismail. 2019. Survey of slam in low-resourced hardware. *IJAIT (International Journal of Applied Information Technology)* 3, 01 (2019), 1–9.

[23] Hanme Jang, Kiyun Yu, and JongHyeon Yang. 2020. Indoor reconstruction from floorplan images with a deep learning approach. *ISPRS International Journal of Geo-Information* 9, 2 (2020), 65.

[24] Chengzu Li, Caiqi Zhang, Han Zhou, Nigel Collier, Anna Korhonen, and Ivan Vulić. 2024. Topviewrs: Vision-language models as top-view spatial reasoners. *arXiv preprint arXiv:2406.02537* (2024).

[25] Ki-Joune Li, Giuseppe Conti, Evdokimos Konstantinidis, Sisi Zlatanova, and Panagiotis Bamidis. 2019. OGC IndoorGML: A standard approach for indoor maps. In *Geographical and Fingerprinting Data to Create Systems for Indoor Positioning and Indoor/Outdoor Navigation*. Elsevier, 187–207.

[26] Chia Xin Liang, Pu Tian, Caitlyn Heqi Yin, Yao Yua, Wei An-Hou, Li Ming, Tianyang Wang, Ziqian Bi, and Ming Liu. 2024. A Comprehensive Survey and Guide to Multimodal Large Language Models in Vision-Language Tasks. *arXiv preprint arXiv:2411.06284* (2024).

[27] Alan M MacEachren. 2004. *How maps work: representation, visualization, and design.* Guilford Press.

[28] Changwei Miao, Guangzhu Chen, Chengliang Yan, and Yuanyuan Wu. 2021. Path planning optimization of indoor mobile robot based on adaptive ant colony algorithm. *Computers & Industrial Engineering* 156 (2021), 107230.

[29] Sherif Mostafa, Khaled A Harras, and Moustafa Youssef. 2023. UniCellular: An Accurate and Ubiquitous Floor Identification System using Single Cell Tower Information. In *ACM SIGSPACIAL*.

[30] Nora Newcombe and Janellen Huttenlocher. 2000. *Making space: The development of spatial representation and reasoning.* MIT Press.

[31] Sebastian Ochmann, Richard Vock, Raoul Wessel, and Reinhard Klein. 2016. Automatic reconstruction of parametric building models from indoor point clouds. *Computers & Graphics* 54 (2016), 94–103.

[32] Brian Okorn, Xuehan Xiong, Burcu Akinci, and Daniel Huber. 2010. Toward automated modeling of floor plans. In *Proceedings of the symposium on 3D data processing, visualization and transmission*, Vol. 2.

[33] Wamiq Para, Paul Guerrero, Tom Kelly, Leonidas J Guibas, and Peter Wonka. 2021. Generative layout modeling using constraint graphs. In *Proceedings of the IEEE/CVF international conference on computer vision*.

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in neural information processing systems*, Vol. 28.

[35] Josselin S Roberts, Tony Lee, Chi H Wong, Michihiro Yasunaga, Yifan Mai, and Percy Liang. 2024. Image2struct: Benchmarking structure extraction for vision-language models. *Advances in Neural Information Processing Systems* 37 (2024), 115058–115097.

[36] Hyung-Gyu Ryoo, Taehoon Kim, and Ki-Joune Li. 2015. Comparison between two OGC standards for indoor space: CityGML and IndoorGML. In *ACM SIGSPATIAL*. 1–8.

[37] Ahmed Saeed, Ahmed Abdelkader, Mouhyemen Khan, Azin Neishaboori, Khaled A Harras, and Amr Mohamed. 2019. On realistic target coverage by autonomous drones. *ACM Transactions on Sensor Networks (TOSN)* 15, 3 (2019), 1–33.

[38] Paul-Edouard Sarlin, Mihai Dusmanu, Johannes L Schönberger, Pablo Speciale, Lukas Gruber, Viktor Larsson, Ondrej Miksik, and Marc Pollefeys. 2022. Lamar: Benchmarking localization and mapping for augmented reality. In *European Conference on Computer Vision*. Springer, 686–704.

[39] Muhammad A Shah, Khaled A Harras, and Bhiksha Raj. 2020. Sherlock: A crowd-sourced system for automatic tagging of indoor floor plans. In *Proc. IEEE 17th Int. Conf. Mobile Ad Hoc Sensor Syst.* 594–602.

[40] Muhammad Ahmed Shah, Bhiksha Raj, and Khaled A Harras. 2017. Inferring room semantics using acoustic monitoring. In *IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE.

[41] Niko Sünderhauf, Feras Dayoub, Sean McMahon, Ben Talbot, Ruth Schulz, Peter Corke, Gordon Wyeth, Ben Upcroft, and Michael Milford. 2016. Place categorization and semantic mapping on a mobile robot. In *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE.

[42] Sebastian Thrun et al. 2002. Robotic mapping: A survey. (2002).

[43] Eric Turner and Avideh Zakhor. 2014. Floor plan generation and room labeling of indoor environments from laser range data. In *2014 international conference on computer graphics theory and applications (GRAPP)*. IEEE, 1–12.

[44] Chaoqun Wang, Jiankun Wang, Chenming Li, Danny Ho, Jiyu Cheng, Tingfang Yan, Lili Meng, and Max Q-H Meng. 2019. Safe and robust mobile robot navigation in uneven indoor environments. *Sensors* 19, 13 (2019), 2993.

[45] Jiayu Wang, Yifei Ming, Zhenmei Shi, Vibhav Vineet, Xin Wang, Sharon Li, and Neel Joshi. 2024. Is a picture worth a thousand words? delving into spatial reasoning for vision language models. *Advances in Neural Information Processing Systems* 37 (2024), 75392–75421.

[46] Sheng Wang, Guohua Gou, Haigang Sui, Yufeng Zhou, Hao Zhang, and Jiajie Li. 2022. Cdsfusion: dense semantic slam for indoor environment using cpu computing. *Remote Sensing* 14, 4 (2022), 979.

[47] Zhongguo Xu, Cheng Yang, Salah Alheejawi, Naresh Jha, Syed Mehadi, and Mrinal Mandal. 2021. Floor plan semantic segmentation using deep learning with boundary attention aggregated mechanism. In *Proceedings of the 2021 4th International Conference on Artificial Intelligence and Pattern Recognition*. 346–353.

[48] Baoding Zhou, Wei Ma, Qingquan Li, Naser El-Sheimy, Qingzhou Mao, You Li, Fuqiang Gu, Lian Huang, and Jiasong Zhu. 2021. Crowdsourcing-based indoor mapping using smartphones: A survey. *ISPRS Journal of Photogrammetry and Remote Sensing* 177 (2021), 131–146.