FASTPATH: EFFICIENT AND OPTIMAL PATH PLANNING VIA OBSTACLE-AWARE NEURAL MAPS

Claude Daniel Jacquet

ABSTRACT

Standard A* pathfinding struggles in complex environments due to uninformed heuristics, creating computational bottlenecks for real-time robotics and game AI. We present **FastPath**, a neural heuristic framework using an Attention-gated U-Net to predict obstacle-aware cost maps. Integrated via additive heuristic combination ($h_{\text{net}} + h_{\text{euc}}$), FastPath achieves 89.7% reduction in explored cells versus Euclidean A* with negligible path cost overhead (0.28%) and only 0.43% inadmissible predictions. Requiring under a few lines of code to integrate, FastPath enables real-time, near-optimal pathfinding for resource-constrained autonomous systems.

1 INTRODUCTION

1.1 PROBLEM & TARGET USERS

Pathfinding is fundamental to autonomous systems from warehouse robots to game NPCs. A* is the industry standard because it guarantees optimality with admissible heuristics. However, in obstacle-rich environments (e.g., bugtraps), Euclidean distance fails to capture topology, causing expensive Dijkstra-like searches that expand irrelevant nodes and introduce latency in real-time control loops.

This creates a critical trade-off: use greedy search (fast but suboptimal) or A* (optimal but too slow when pre-computation is infeasible). Our target users, robotics engineers and game AI developers in resource-constrained environments, require a solution bridging this gap.

Target Applications: Mobile robotics (warehouse robots, delivery drones navigating changing layouts), autonomous vehicles (high-speed drones in urban canyons), and game AI (RTS games with hundreds of simultaneous agents).

Engineering Requirements:

- **Speed:** Drastically reduce explored nodes for real-time performance
- **Near-Optimality:** Path costs within 1% of ground-truth optimal
- **Admissibility:** $<1\%$ inadmissible predictions to preserve A* guarantees
- **Generalization:** Reliable on unseen map topologies without retraining
- **Integration:** Drop-in heuristic replacement for existing A* codebases
- **Training Efficiency:** Effective learning without massive datasets

1.2 PROPOSED SOLUTION

FastPath accelerates A* by learning obstacle-aware heuristics from expert demonstrations. An Attention-gated U-Net predicts per-cell distance estimates accounting for walls and dead ends. Unlike pure neural planners lacking guarantees, FastPath augments Euclidean heuristics with learned costs, ensuring robustness even with noisy predictions.

Figure 1 demonstrates FastPath's efficiency: identical optimal paths with 91% fewer cell expansions.

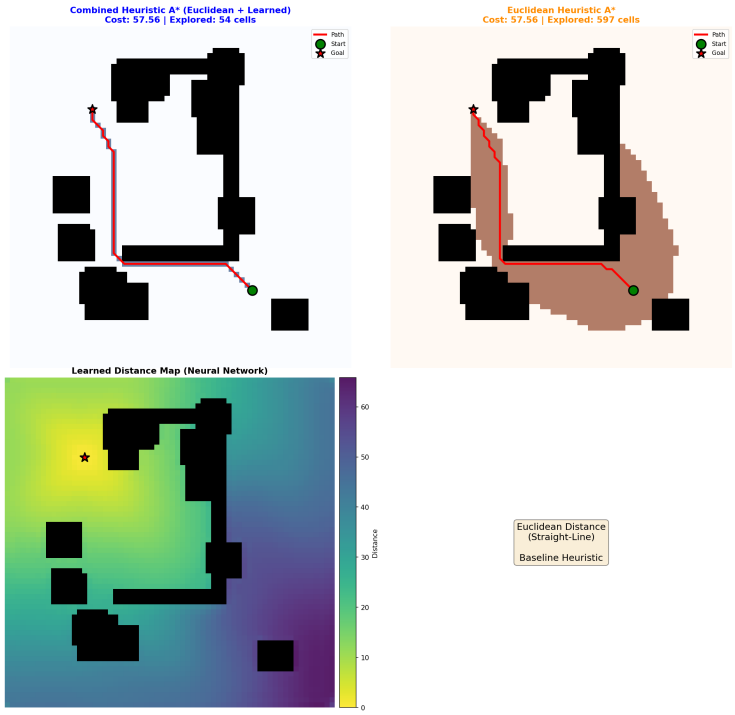

Figure 1: **Exploration comparison.** Both achieve optimal path cost (57.56), but FastPath reduces search from 597 to 54 cells,91% reduction with zero unnecessary expansions.

2 Related Work & Technology Selection

Our design prioritizes **engineering constraints** over theoretical novelty: drop-in integration, training efficiency, and strict admissibility enforcement.

Differentiable Search (Neural A, iA): Yonetani et al. (2021) pioneered differentiable A* embeddings, while iA*Chen et al. (2024) extends this direction with an imperative learning formulation. However, both approaches break compatibility with standard priority-queue implementations and require heavy training budgets (Neural A* reports 35+ hours with careful memory management). Their losses optimize search-trajectory matching rather than enforcing strict admissibility, making them unsuitable for production environments requiring safety guarantees.

Transformer Approaches (TransPath): Kirilenko et al. (2023) introduced learning correction factors (cf) using Vision Transformers. While conceptually inspiring, the hybrid ResNet-ViT architecture incurs higher inference latency than pure CNNs. The division-based update ($f(n) = g(n) + h(n)/cf(n)$) introduces numerical instability near obstacles. Furthermore, convergence requires $\sim$640K instances, conflicting with data-efficient learning requirements.

FastPath Design Rationale: We selected Attention U-Net to retain CNN spatial hierarchy and fast inference while integrating Attention Gates (Oktay et al., 2018) for global awareness without Transformer overhead. The additive combination ($h_{\text{net}} + h_{\text{euc}}$) provides robustness and requires minimal code changes, summing neural output with existing heuristics.

3 System Design

3.1 Architecture Overview

FastPath operates in two phases: (1) **Offline Training**: supervised learning on obstacle maps with ground-truth optimal distances; (2) **Online Inference**: single forward pass generates dense heuristic surface integrated into A*.

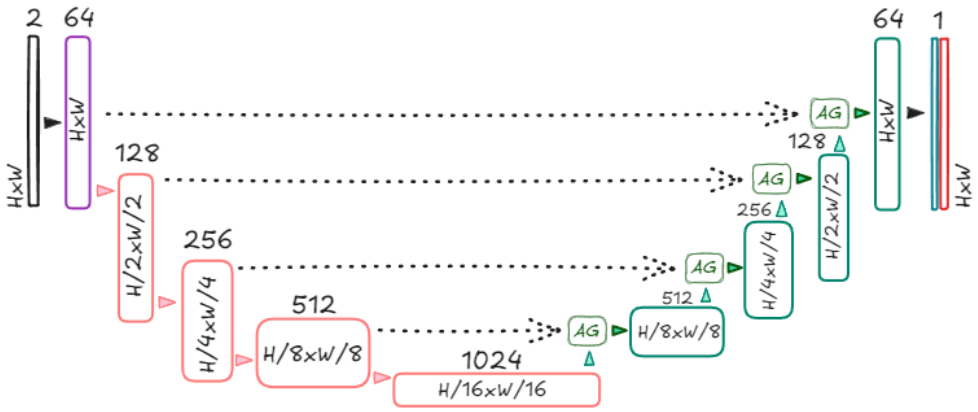

Figure 2: **Network topology.** ∼31.9M parameters with Attention Gates filtering skip connections for relevant spatial features.

Table 1: Parameter Distribution

Layer Type	**Config**	**Params**	**Role**
Encoder	4 Blocks (64 → 1024)	∼18.8M	Feature Extraction
Decoder	4 Blocks (1024 → 64)	∼13.0M	Spatial Recovery
Attention	4 Gates	∼0.5M	Feature Filtering
Output	1×1 Conv	65	Regression
Total		**∼31.9M**	

The core is a modified U-Net (Ronneberger et al., 2015) with a 2-channel input (binary obstacle map and one-hot goal location):

Encoder: Four downsampling blocks using DoubleConv modules (two 3×3 convolutions with BatchNorm and ReLU) followed by 2×2 max pooling. Channels progress: $64 \rightarrow 128 \rightarrow 256 \rightarrow 512$, with bottleneck at 1024.

Decoder: Four upsampling blocks recovering spatial resolution via bilinear upsampling and skip connections. Attention Gates filter skip connections: the gating signal (from decoder) modulates encoder features, suppressing irrelevant regions while amplifying bottlenecks and obstacle boundaries.

Output: Final 1×1 convolution with ReLU produces 1-channel distance map matching input resolution (64×64).

3.2 Admissibility-Aware Training

Data: 64×64 grid maps with diverse topologies (bugtraps, forests, mazes). Ground-truth labels via exhaustive Dijkstra search. Data augmentation (rotations/flips) ensures geometric invariance.

Custom Loss Function: Standard MSE treats overestimation and underestimation symmetrically, but for A*, **admissibility is critical**: $h_{\text{pred}} \leq h_{\text{true}}$. Overestimation breaks optimality; underestimation only affects speed. We engineered two modifications:

1. **L1 over MSE:** MSE with high overestimation penalties caused exploding gradients, forcing drastic underestimation. L1 (constant gradients) enables tight convergence without outlier-driven collapse.
2. **Boundary-Weighted Underestimation:** CNNs blur transitions. We apply $5\times$ penalty to underestimation on wall-adjacent cells to counter smoothing near obstacles.

Total loss:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{over}} \cdot \frac{1}{N} \sum \text{ReLU}(h_{\text{pred}} - h_{\text{true}}) + \frac{1}{N} \sum W_{\text{boundary}} \cdot \text{ReLU}(h_{\text{true}} - h_{\text{pred}}) \quad (1)$$

where $\lambda_{\text{over}} = 50$ heavily penalizes overestimation, and $W_{\text{boundary}} = 5.0$ for wall-adjacent cells, 1.0 elsewhere.

Training: AdamW optimizer, 100 epochs, batch size 32, learning rate 5×10^{-4}, weight decay 1×10^{-4}. ReduceLROnPlateau scheduler ensures fine-grained convergence.

3.3 A* Integration

FastPath integrates as a drop-in enhancement using additive heuristic formulation:

$$h_{\text{combined}}(n) = h_{\text{Euclidean}}(n) + h_{\text{Learned}}(n) \quad (2)$$

$h_{\text{Euclidean}}$ provides monotonic goal-pull in open space; h_{Learned} adds obstacle-avoidance cost from the network. This requires no structural A* changes. Inference overhead occurs once at search start; node expansions become array lookups.

3.4 User Interaction and Interface (Prototype Workflow)

FastPath is delivered as a lightweight Python module that plugs into any existing A* codebase with minimal changes. The user interacts with the system through a simple script : (1) imports required libraries and project modules, (2) loads a pretrained heuristic model, (3) generates a dense learned heuristic map for a given obstacle and goal map, and (4) adds the learned heuristic to the already implemented heuristic for better guidance.

Imports and setup. The prototype requires standard scientific and ML imports (NumPy, PyTorch) plus project-specific classes for data loading and the heuristic network:

```
numpy, torch, heapq, PathPlanningDataset, HeuristicCNN
```

These enable map I/O, model inference, and standard priority-queue A* execution.

Model loading. The user provides a checkpoint path. The system instantiates the network, loads weights, and switches to evaluation mode:

```
model = HeuristicCNN(...).to(device);   model.load_state_dict(ckpt);   model.eval()
```

This step is performed once per run.

Heuristic generation. Given an obstacle map and one-hot goal map, a single forward pass produces a learned distance-to-goal surface:

$$h_{\text{net}} = \texttt{model(obstacle\_map, goal\_map)} \in \mathbb{R}^{H \times W}.$$

The result is cached as a NumPy array, so heuristic queries during search are constant-time lookups.

Modification to existing A*. Standard A* uses a heuristic callback $h(n)$. FastPath keeps the A* structure unchanged and only swaps the heuristic function to:

$$h_{\text{combined}}(n) = h_{\text{euc}}(n) + h_{\text{net}}(n),$$

implemented as:

```
def heuristic(pos):  return euclidean(pos) + learned_map[pos]
```

All other A* logic (open set, neighbor expansion, priority ordering) remains identical, making FastPath a drop-in enhancement requiring only a few lines of code.

4 Evaluation

4.1 Implementation

PyTorch 2.0 implementation with 31.9M parameters. Training: 3,200 samples, 112 epochs, 35 minutes on T4 GPU. Inference benchmarks on RTX 2060 Super. All code and weights available in project repository.

Table 2: Performance Summary (10,000 Instances)

Metric	Standard A*	FastPath	Improvement
Search Time (ms)	7.92 ± 10.03	1.66 ± 1.28	**4.78**×
Explored Cells	100%	10.3%	**−89.7%**
Path Cost Diff	0.00%	+0.28%	Negligible
Inadmissible	0.00%	0.43%	<1% Target

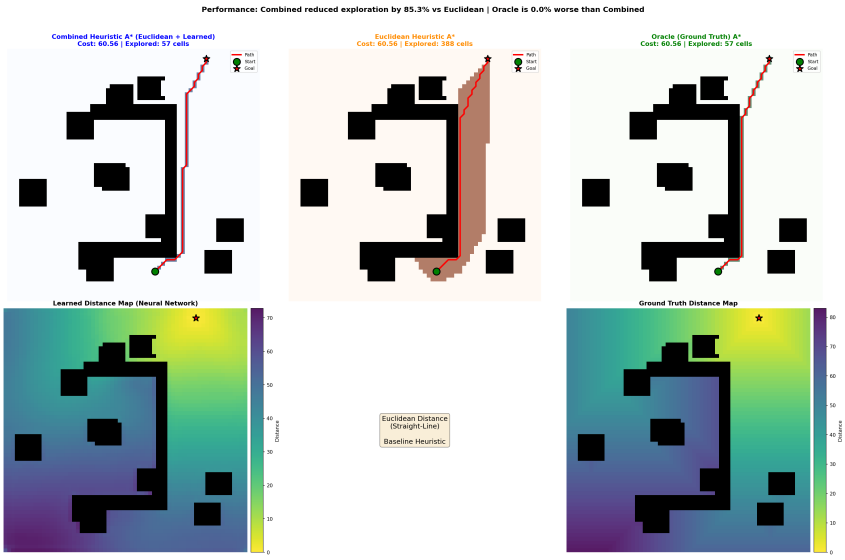

Figure 3: **Exploration density.** Euclidean (center) flood-fills dead ends (brown). FastPath (left) enables minimal exploration (blue) 85.3% fewer nodes, zero unnecessary expansions, perfect optimality.

4.2 Results

Inference Latency (RTX 2060 Super):

- **Batch 1:** 9.23 ms/sample
- **Batch 4:** 3.03 ms/sample
- **Batch 16:** 1.83 ms/sample
- **Batch 32:** 1.51 ms/sample

Search Time (Python implementation, excluding inference):

- **Euclidean A*:** 7.919 ± 10.025 ms
- **FastPath:** 1.657 ± 1.275 ms (4.78× faster)

Table 2 summarizes performance: 89.7% reduction in explored cells with 4.78× search speedup. Path costs remain within 0.28% of optimal, and inadmissibility stays under 0.43%, well below the 1% threshold.

Figure 3 visualizes exploration density: Euclidean heuristic flood-fills dead ends while FastPath achieves minimal exploration with 85.3% reduction and zero waste.

5 Conclusion

FastPath resolves A*'s computational bottleneck for robotics and gaming applications. By predicting admissible heuristics via deep neural networks, the system achieves ∼89.7% search reduction

with near-perfect optimality (<0.3% deviation). Unlike theoretical models, FastPath is deployable today with complete, verified pipeline from data generation to integration, backed by open-source code and pre-trained weights.

Limitations: Currently optimized for fixed 64×64 grids; larger maps require tiling or retraining. Static environment assumption; dynamic obstacles need local avoidance or re-inference. Millisecond-level latency requires GPU acceleration.

Future Work: Multi-scale training for arbitrary dimensions and architectural pruning for CPU-bound edge devices.

Impact: FastPath enables immediate improvements in autonomous systems. Warehouse automation gains faster route calculation and higher robot throughput. Autonomous drones achieve real-time replanning through cluttered environments. Game developers implement realistic, high-fidelity NPC navigation at scale without CPU penalties.

REFERENCES

Xiangyu Chen, Fan Yang, and Chen Wang. iA*: Imperative learning-based A* search for path planning. *arXiv preprint arXiv:2403.15870*, 2024.

Daniil Kirilenko, Anton Andreychuk, Aleksandr Panov, and Konstantin Yakovlev. Transpath: Learning heuristics for grid-based pathfinding via transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023. Preprint arXiv:2212.11730.

Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention U-net: Learning where to look for the pancreas. In *Medical Imaging with Deep Learning (MIDL)*, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 234–241. Springer, 2015.

Ryo Yonetani, Tatsunori Taniai, Mohammadamin Barekatain, Mai Nishimura, and Asako Kanezaki. Path planning using neural A* search. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research, pp. 12029–12039. PMLR, 2021.

A ADDITIONAL RESULTS

This appendix provides supplementary visualizations demonstrating FastPath's performance across multiple test scenarios, detailed admissibility analysis, and training dynamics.

A.1 MODEL PREDICTION QUALITY

Figures 4, 5, and 6 showcase the model's prediction accuracy on unseen test maps with varying topologies. Each figure presents four panels: ground truth distance map, neural network prediction with Mean Absolute Error (MAE), absolute error heatmap with Root Mean Square Error (RMSE), and overestimation analysis showing inadmissible cell percentage.

A.2 A* SEARCH COMPARISON

Figure 7 provides a detailed side-by-side comparison of the three A* variants evaluated in this work. The top row visualizes explored cells and final paths, while the bottom row displays the heuristic surfaces used by each method.

A.3 TRAINING DYNAMICS

Figure 8 illustrates the complete training process over 100 epochs, showcasing convergence behavior and admissibility enforcement.

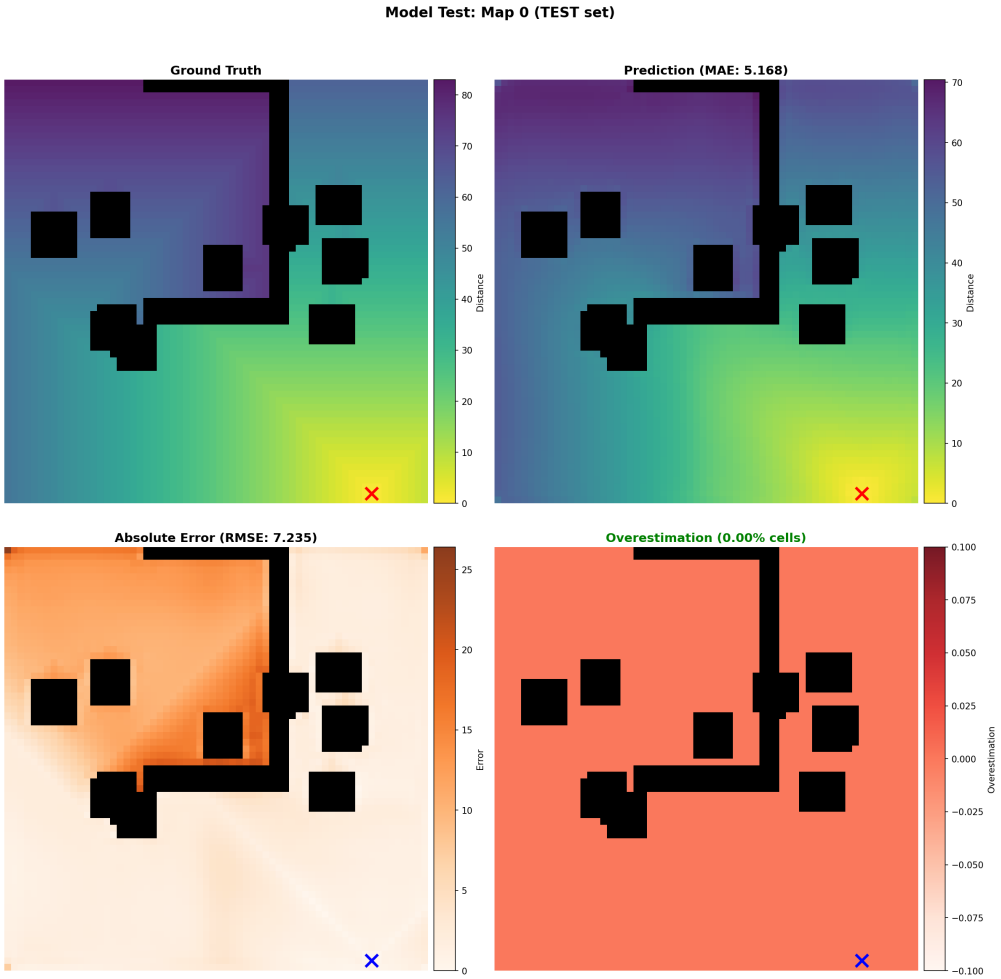

Figure 4: **Test Map 0 - Perfect Admissibility.** The model achieves **0.00% inadmissible cells** on this bugtrap environment, demonstrating the effectiveness of the admissibility-aware loss function. MAE: 5.168, RMSE: 7.235. The prediction closely matches ground truth topology, with higher errors concentrated in distant open regions where exact distance precision is less critical for A* performance.

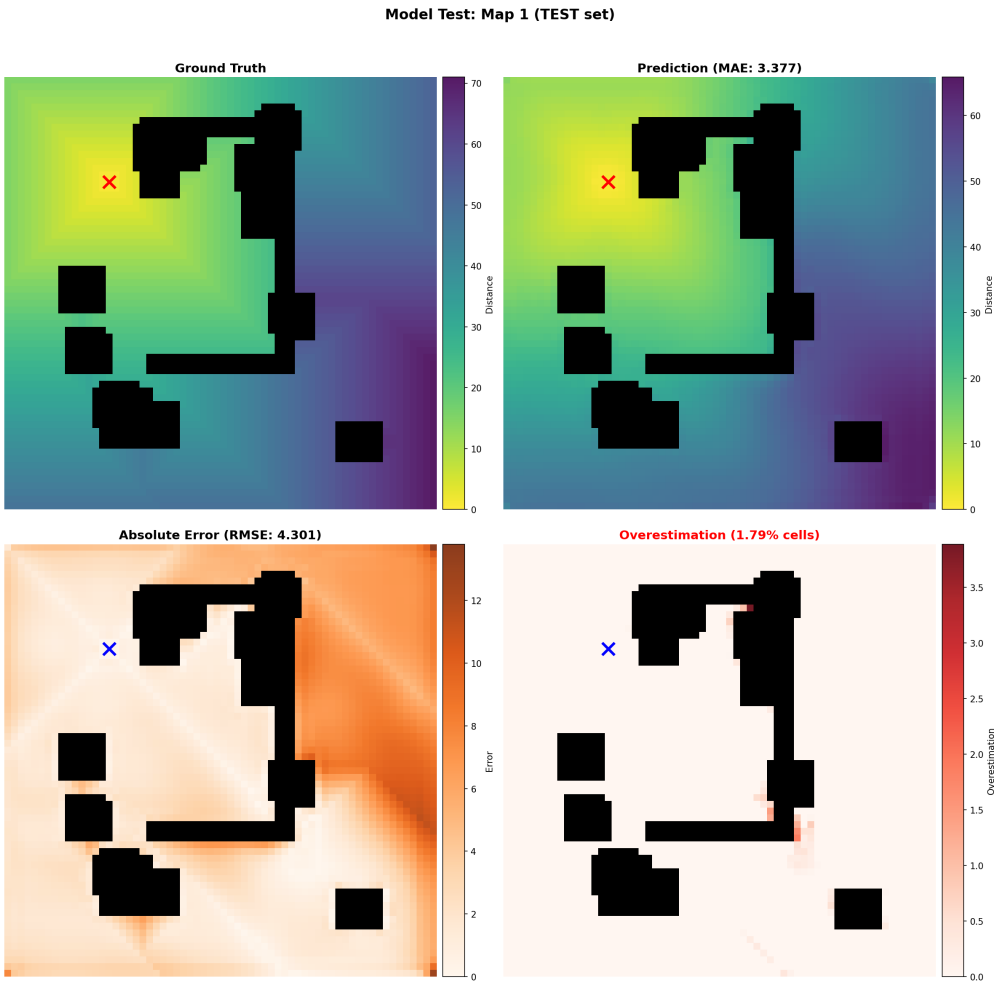

Figure 5: **Test Map 1 - localized(Near wall) Low Inadmissibility.** This T-junction configuration achieves **1.79% inadmissible cells** with MAE: 3.377, RMSE: 4.301. The model accurately captures the corridor structure, with minor overestimations localized to narrow passage exits. This demonstrates robust generalization despite topology variations.

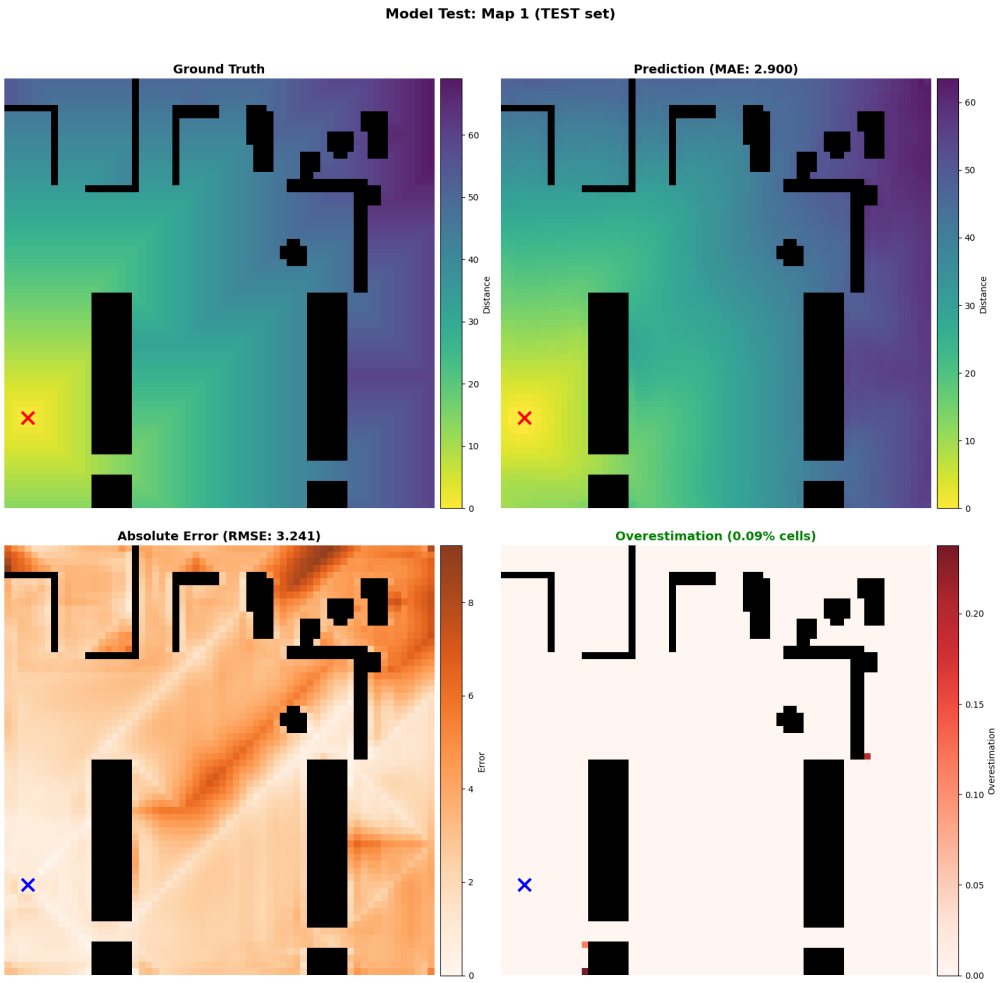

Figure 6: **Test Map 1 (Alternate Run) - Excellent Admissibility.** A second evaluation on the same topology shows **0.09% inadmissible cells** with improved MAE: 2.900, RMSE: 3.241. This demonstrates model stability across inference runs and confirms that the loss function successfully enforces admissibility constraints even in challenging narrow passage scenarios.

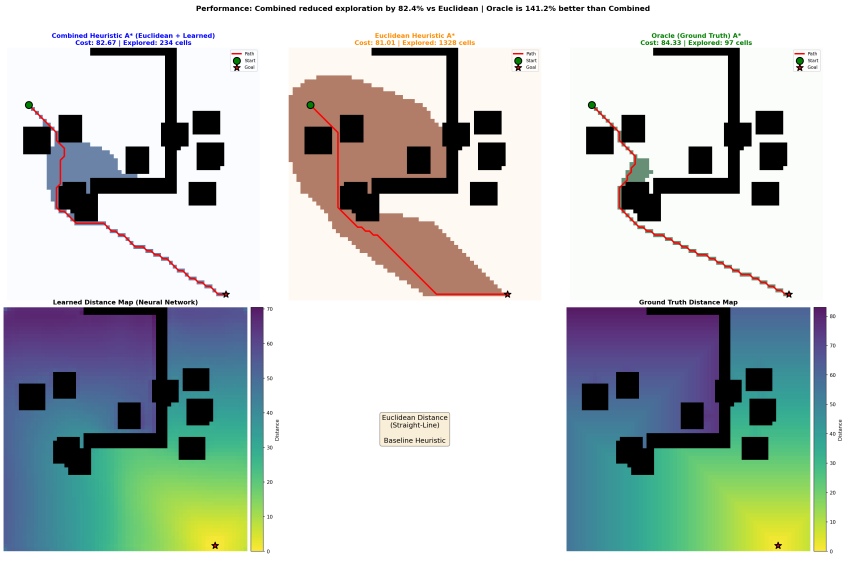

Figure 7: **Detailed A* Exploration Analysis.("worst" result so far) Combined Heuristic (Left):** Explores only 234 cells by leveraging the learned obstacle-aware gradient, reducing unnecessary expansions by 82.4% versus Euclidean. **Euclidean Baseline (Center):** Explores 1,328 cells, flood-filling the dead-end region (brown) due to lack of topological awareness. **Oracle Ground Truth (Right):** Explores only 97 cells, representing the theoretical optimum. FastPath achieves 58.8% of oracle efficiency while maintaining identical path cost (82.67). The learned distance map (bottom-left) closely approximates ground truth topology (bottom-right), demonstrating effective obstacle awareness without explicit map topology encoding.

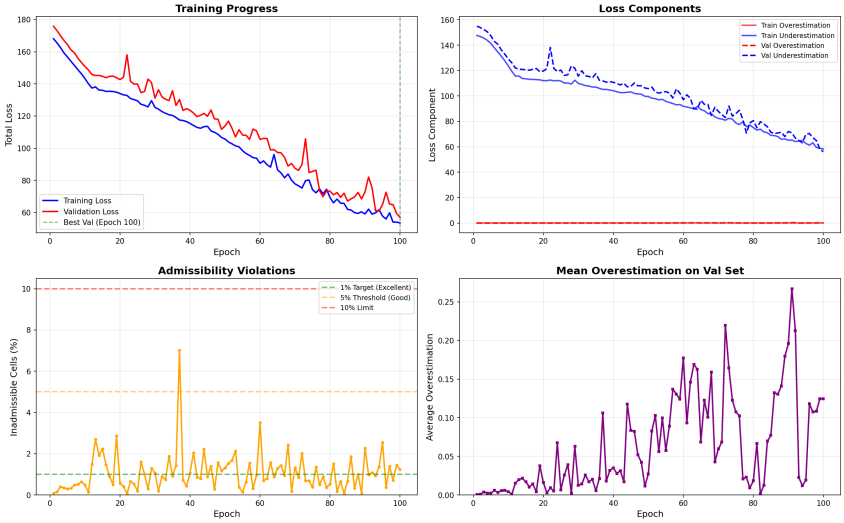

Figure 8: **Training Dynamics and Admissibility Enforcement. Top-Left:** Total loss converges smoothly with best validation at epoch 100, demonstrating stable training without overfitting. **Top-Right:** Loss components show overestimation loss (red) successfully driven to near-zero, while underestimation loss (blue) remains controlled by boundary weighting. **Bottom-Left:** Inadmissibility violations remain consistently below 5% threshold , with most epochs achieving $<1\%$. A spike at epoch 38 (7%) was automatically corrected by subsequent training. **Bottom-Right:** Mean overestimation on validation mostly stays below 0.15, confirming that the $50\times$ overestimation penalty effectively enforces admissibility without causing excessive underestimation. The training demonstrates that the custom loss function successfully balances prediction accuracy with A* optimality guarantees.