Path Planning using Neural A* Search (Supplementary Material)

Ryo Yonetani [*1] **Tatsunori Taniai** [*1] **Mohammadamin Barekatain** [1 2] **Mai Nishimura** [1] **Asako Kanezaki** [3]

A. Details of Experimental Setups

A.1. Dataset Creation

Here we present supplementary information on how each dataset was created in our experiments. Since our dataset generation process involves randomness, we fixed a random seed in each generation program to ensure reproducibility. Please refer to the code in our project page: https://omron-sinicx.github.io/neural-astar/.

MP/Tiled-MP/CSM datasets. In the experiments with MP/Tiled-MP/CSM datasets, we employed more challenging settings involving randomized start and goal locations instead of pre-defined consistent locations used in prior work (Choudhury et al., 2018; Vlastelica et al., 2020). We determined these start and goal locations strategically based on their actual distances to avoid generating easy problem instances. Specifically, for each environmental map, a single goal location was randomly determined once and fixed throughout the experiments. Here, for a map with the width and height denoted as (W, H), *i.e.*, $(32, 32)$ for the MP and $(64, 64)$ for the Tiled MP and CSM datasets, the goal location was sampled from one of the four corner regions of size $(W/4, H/4)$, as illustrated in Fig. S1 (middle). Then, we performed the Dijkstra algorithm to compute actual movement costs from every passable node to the goal, and calculated the $55, 70, 85$-percentile points of the costs. For every iteration in the training phase, we sampled a new random start location from regions whose actual costs higher than the 55 percentile point. As for validation and testing data, we sampled two and five random but fixed start locations, respectively, from each of three kinds of regions whose costs are within the percentile ranges of $[55, 70]$, $[70, 85]$, and $[85, 100]$ (see Fig. S1 (right) for an illustration). Consequently, we created $2 \times 3 = 6$ and $5 \times 3 = 15$ diverse start locations for each validation and test map, respectively. The ground-truth paths for all the generated problem instances were obtained by the Dijkstra algorithm. When computing losses for the Tiled MP and CSM datasets, these paths were dilated with a 3×3 kernel, which stabilized the training.

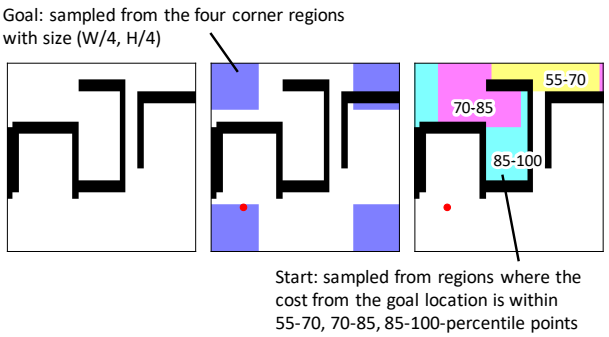

Figure S1. **Sampling of Start and Goal Locations**.

SDD. In SDD, we extracted relatively simple trajectories of pedestrians who moved directly towards their destinations. Specifically, for each trajectory provided by Robicquet et al. (2016), we first trimmed it randomly to have a sequence length in the range of $[300, 600]$ timesteps (at 2.5fps). We then calculated the ratio of its straight-line distance between start and goal points to the trajectory length, as a measure of path simplicity that gives a lower value for a more complex trajectory. We discarded trajectories whose simplicity was less than 0.5. Finally, we cropped image patches that encompass each trajectory with the margin of 50 pixels and resized them to the size of 64×64. As a result, 8,325 samples were extracted from the dataset.

A.2. Hyper-Parameter Selection

Table S1 shows the list of hyper-parameters as well as ranges of these values we tried to produce the final results. We selected the final parameters based on the validation performance on the MP dataset in terms of Hmean scores. Because completing all the experiments took a considerably long time (see the next section), we performed each experiment only once with a fixed set of random seeds.

Below we provide several remarks regarding the hyper-

[*]Equal contribution [1]OMRON SINIC X, Tokyo, Japan [2]Now at DeepMind, London, UK. [3]Tokyo Institute of Technology, Tokyo, Japan. Correspondence to: Ryo Yonetani <ryo.yonetani@sinicx.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

Table S1. **Hyper-parameter Selection.** The list of hyper-parameters and ranges of these values tried during the development of this paper.

Parameter Name	Values (range of values tried)
Common	
Optimizer	RMSProp
Learning rate	0.001 (0.0001, 0.0005, 0.001, 0.005)
Batch size	100
Number of epochs	100 (MP), 400 (Tiled MP, CSM)
Tie breaking (for h func.)	$0.001\times$ Euclidean distance
Neural A*/Neural BF	
Encoder arch	U-Net with VGG-16 backbone (VGG-16, ResNet-18)
Temperature τ	$\sqrt{32}$ for MP, $\sqrt{64}$ for Tiled MP and CSM
BB-A*	
Encoder arch	U-Net with VGG-16 backbone
Trade-off parameter λ	20.0 (0.1, 1.0, 5.0, 10.0, 20.0)
SAIL/SAIL-SL	
Max data samples	300 (60, 300)
Sampling rate	10 (10, 100)
WA*	
Weight factor for $h(v)$	0.8 (0.6, 0.7, 0.8)

parameter list. We observed that the tie-breaking in A* search, *i.e.*, the adjustment to $h(v)$ by adding the Euclidean distance from v to the goal scaled by a small positive constant (0.001), was critical to improving the base efficiency of A* search. Thus, we used this setting for all the A*-based methods throughout the experiments. The choice of the learning rate little affected final performances given a sufficient number of epochs. BB-A* has an additional hyper-parameter λ that controls the trade-off between "informativeness of the gradient" and "faithfulness to the original function" (Vlastelica et al., 2020). We tried several values and found that any choice worked reasonably well, except for extremely small values (*e.g.*, 0.1). SAIL and SAIL-SL (Choudhury et al., 2018) have hyper-parameters on how to collect samples from each training environment instance, which little affected final performances. Finally, the weighted A* baseline used a modified node selection criterion with a weight factor w to the heuristic function; *i.e.*, $(1-w)\cdot g(v) + w \cdot h(v)$, for which we set $w = 0.8$ throughout the experiments. Note that using $w = 1.0$ for the criterion corresponds to the best-first search in our baselines.

A.3. Computing Infrastructure and Training Time

We performed all the experiments on a server operated on Ubuntu 18.04.3 LTS with NVIDIA V100 GPUs, Intel Xeon Gold 6252 CPU @ 2.10GHz (48 cores), and 768GB main memory. Our implementation was based on PyTorch 1.5 (Paszke et al., 2019) and Segmentation Models PyTorch (Yakubovskiy, 2020). To efficiently carry out the experiments, we used GNU Parallel (Tange, 2011) to run multiple experiment sessions in parallel. See our `setup.py` and `Dockerfile` for the list of all dependencies.

In the training sessions, each model was trained using a single V100 GPU with 16 GB graphics memory. Training each of our models (Neural A* and Neural BF) took approximately 50 minutes on the MP dataset (100 epochs $\times$ 800 maps with the size of 32×32) and 35 hours on the Tiled MP and CSM datasets (400 epochs $\times$ 3,200 maps with the size of 64×64). As for SAIL/SAIL-SL and BB-A*, the training on the MP dataset took approximately the same time (*i.e.*, 50 minutes). On the Tiled MP and CSM datasets, SAIL/SAIL-SL took up to about 22 hours and BB-A* took 65 hours.

B. Additional Results

Figures S2 and S3 show additional qualitative results (including those of MMP introduced below.) In what follows, we also add more detailed performance analysis to Neural A* from different perspectives.

B.1. Comparisons with Imitation Learning Methods

As introduced in Sec. 6, some of the imitation learning (IL) methods are relevant to our work in that they learn to recover unknown reward (*i.e.*, negative cost) functions from demonstrations. Here, we compared our approach with several IL methods tailored to path planning tasks, namely, Maximum Margin Planning (MMP) (Ratliff et al., 2006), Value Iteration Network (VIN) (Tamar et al., 2016) and Gated Path Planning Network (GPPN) (Lee et al., 2018).

For MMP, we followed the original work and modeled the cost function as a per-node linear mapping from features to costs. For feature extraction, we used the VGG-16 network pretrained on ImageNet. As in Neural A*, we activated the costs with the sigmoid function to constrain them to be in $[0, 1]$. Unlike other IL-based planners, MMP uses A* search to plan a path with the estimated cost. We used the same implementation of A* search as that of Neural A*.

For VIN and GPPN, we used the official codebase of Lee et al. (2018). Because these reactive planners are not based on a search-based algorithm, we could not employ the Exp and Hmean metrics, which are associated with performances of the baseline A* search. Instead, we calculated the success ratio (Suc), which is the percentage of problem instances for which a planner found a valid path.

As shown in Tables S2 and S3, we confirmed that the performances of these IL methods are limited compared to the proposed Neural A*. Although MMP ensures 100% planning success as using A* search, it was consistently outperformed by Neural A* in terms of the Opt, Exp, and Hmean metrics. One possible reason of these limited performances is that

MMP cannot learn how internal search steps of A* affect search histories and resulting paths, as we compared BB-A* against Neural A* in Sec. 4.4. While GPPN obtained a higher Opt score than Neural A* on the Tiled MP dataset, it did not always successfully find a valid path as shown in its success ratio. Moreover, GPPN and VIN completely failed to learn on SDD. These failures can be accounted for by its large input maps and limited demonstrations (single trajectory per map), which are more challenging settings than those by the original work.

B.2. Path Length Optimality Evaluation

Following Tamar et al. (2016); Anderson et al. (2018), we introduce another metric for evaluating the path optimality, which calculates the ratio of the optimal path length $\bar{P}$ to predicted one P. For consistency with the other metrics, this path-length ratio is measured in 0–100 (%) and maximized when the path is optimal, *i.e.*, we calculate $|\bar{P}|/|P| \times 100$. As shown in Table S4, Neural A* produced the most nearly-optimal paths across all the datasets.

B.3. Computational Complexity and Runtime Analysis

The main computational bottleneck of Neural A* lies in the differentiable A* module. Because this module uses matrix operations involving all the nodes to enable back-propagation, its training-phase complexities in space and time are $\mathcal{O}(k|\mathcal{V}|)$, where k is the number of search steps. In theory, the required number of search steps depends on the true path length d. Thus, the complexities in terms of d amount to $\mathcal{O}(d|\mathcal{V}|)$ and $\mathcal{O}(b^d|\mathcal{V}|)$ for the best and worst case, respectively, where b is the average number of neighboring nodes expanded per search step. Practically, when training is finished, we can replace the differentiable A* module with a standard (*i.e.*, non-differentiable) A* search algorithm without changing the testing behaviors of Neural A*. With this replacement, Neural A* can perform planning in $\mathcal{O}(d)$ and $\mathcal{O}(b^d)$ for the best and worst case.

To analyze search runtimes empirically, we created three sets of 50 maps with the sizes of 64×64, 128×128, and 256×256, by randomly drawing maps from the MP dataset and tiling them. We then measured the average runtime per problem taken using a standard A* search implementation[1] coupled with and without our guidance maps. Regardless of the test map sizes, the guidance maps were trained using the Tiled MP dataset of the size 64×64, to see if our model generalizes well to larger maps. As shown in Table S5, we confirm that Neural A* greatly reduced runtimes of A* search with the help of guidance maps and also showed good generalization ability to larger maps.

[1] We implemented a python-based A* search algorithm with `pqdict` package (https://github.com/nvictus/priority-queue-dictionary) and used its priority queue feature for performing node selections efficiently. To accurately measure a search runtime per problem, we performed the program exclusively on a single CPU core (for performing A* search) and a single GPU (for additionally running the encoder to compute guidance maps for Neural A*) and solved the same problem five times after a single warm-up trial. The use of more sophisticated A* search implementations could result in further performance improvements, which is however beyond the scope of this work.

References

Anderson, P., Chang, A., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

Choudhury, S., Bhardwaj, M., Arora, S., Kapoor, A., Ranade, G., Scherer, S., and Dey, D. Data-driven planning via imitation learning. *International Journal of Robotics Research (IJRR)*, 37 (13-14):1632–1672, 2018.

Lee, L., Parisotto, E., Chaplot, D. S., Xing, E., and Salakhutdinov, R. Gated path planning networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8024–8035, 2019.

Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. Maximum margin planning. In *Proceedings of the International Conference on Machine Learning (ICML)*, ICML '06, pp. 729–736, 2006.

Robicquet, A., Sadeghian, A., Alahi, A., and Savarese, S. Learning social etiquette: Human trajectory understanding in crowded scenes. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 549–565, 2016.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. Value iteration networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2154–2162, 2016.

Tange, O. Gnu parallel - the command-line power tool. *The USENIX Magazine*, 36(1):42–47, Feb 2011. URL http://www.gnu.org/s/parallel.

Vlastelica, M., Paulus, A., Musil, V., Martius, G., and Rolínek, M. Differentiation of blackbox combinatorial solvers. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Yakubovskiy, P. Segmentation models pytorch. https://github.com/qubvel/segmentation_models.pytorch, 2020.

Table S2. **Comparisons with Imitation Learning Methods.** Bootstrap means and 95% confidence bounds of path optimality ratio (Opt), reduction ratio of node explorations (Exp), the harmonic mean (Hmean) between Opt and Exp, and success ratio (Suc).

MP Dataset				
	Opt	Exp	Hmean	Suc
VIN	24.7 (23.9, 25.5)	N/A	N/A	31.4 (30.6, 32.3)
GPPN	71.0 (70.2, 71.8)	N/A	N/A	86.2 (85.6, 86.9)
MMP	81.6 (80.6, 82.8)	22.4 (21.7, 23.2)	31.5 (30.6, 32.4)	**100.0 (100.0, 100.0)**
Neural A*	**87.7 (86.6, 88.9)**	**40.1 (38.9, 41.3)**	**52.0 (50.7, 53.3)**	**100.0 (100.0, 100.0)**
Tiled MP Dataset				
	Opt	Exp	Hmean	Suc
VIN	52.7 (51.5, 54.0)	N/A	N/A	58.3 (57.1, 59.5)
GPPN	**68.2 (67.0, 69.4)**	N/A	N/A	81.5 (80.5, 82.5)
MMP	44.8 (42.4, 47.1)	40.5 (38.7, 42.4)	35.5 (33.8, 37.2)	**100.0 (100.0, 100.0)**
Neural A*	63.0 (60.7, 65.2)	**55.8 (54.1, 57.5)**	**54.2 (52.6, 55.8)**	**100.0 (100.0, 100.0)**
CSM Dataset				
	Opt	Exp	Hmean	Suc
VIN	70.4 (69.3, 71.6)	N/A	N/A	73.2 (72.1, 74.4)
GPPN	68.9 (67.7 60.1)	N/A	N/A	85.3 (84.4, 86.2)
MMP	66.4 (64.0, 68.9)	28.4 (26.4, 30.4)	31.9 (30.0, 33.8)	**100.0 (100.0, 100.0)**
Neural A*	**73.5 (71.5, 75.5)**	**37.6 (35.5, 39.7)**	**43.6 (41.7, 45.5)**	**100.0 (100.0, 100.0)**

Table S3. **Comparisons with Imitation Learning Methods on SDD.** Bootstrap means and 95% confidence bounds of the chamfer distance between predicted and actual pedestrian trajectories.

	Intra-scenes	Inter-scenes
VIN	920.7 (890.8, 950.4)	900.6 (888.1, 913.8)
GPPN	920.7 (890.8, 952.3)	900.6 (888.0, 913.4)
MMP	126.7 (119.3, 133.9)	130.3 (128.1, 132.6)
Neural A*	**16.1 (13.2, 18.8)**	**37.4 (35.8, 39.0)**

Table S4. **Path Length Optimality Evaluation.** Bootstrap means and 95% confidence bounds of the ratio of optimal to produced path lengths (the higher the better).

	MP	Tiled-MP	CSM
BF	96.4 (96.1, 96.6)	92.1 (91.6, 92.6)	96.4 (96.1, 96.7)
WA*	96.9 (96.6, 97.1)	93.4 (93.0, 93.8)	96.8 (96.6, 97.1)
SAIL	87.5 (86.8, 88.3)	78.0 (77.0, 79.0)	87.1 (86.1, 88.1)
SAIL-SL	88.2 (87.5, 89.0)	73.4 (72.1, 74.7)	84.7 (83.5, 85.9)
BB-A*	96.3 (96.0, 96.6)	93.0 (92.5, 93.4)	96.5 (96.2, 96.8)
Neural BF	97.5 (97.3, 97.8)	95.0 (94.7, 95.4)	97.4 (97.1, 97.6)
Neural A*	**99.1 (99.0, 99.2)**	**98.4 (98.3, 98.6)**	**98.9 (98.8, 99.0)**

Table S5. **Search Runtime Evaluation.** Bootstrap mean and 95% confidence bounds of the runtime (sec) required to solve a single problem with different map sizes.

	64×64	128×128	256×256
A*	0.09 (0.08, 0.10)	0.21 (0.17, 0.25)	0.78 (0.72, 0.82)
Neural A*	**0.04 (0.04, 0.04)**	**0.07 (0.06, 0.08)**	**0.15 (0.14, 0.16)**

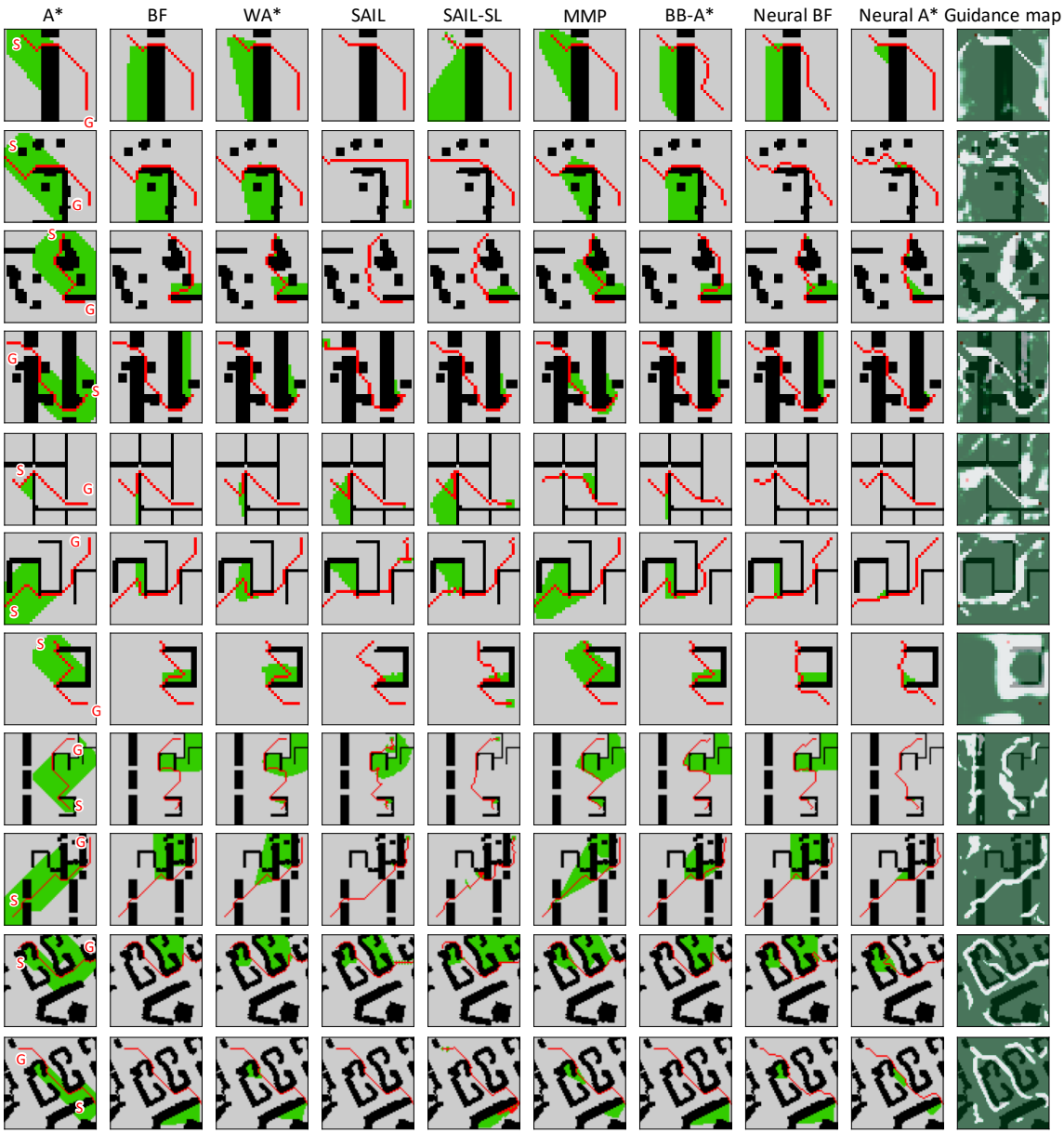

Figure S2. **Additional Qualitative Results (MP/Tiled-MP/CSM Datasets).**

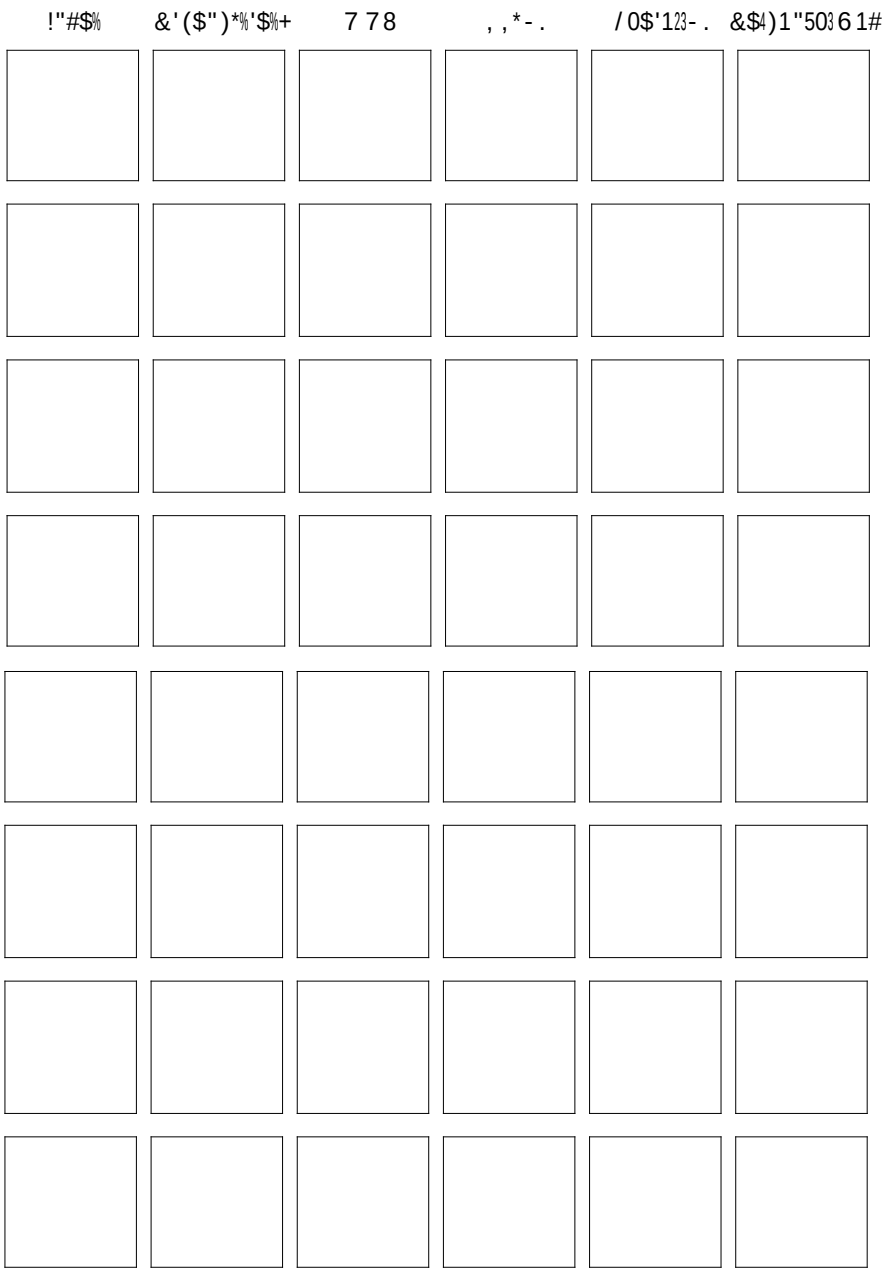

Figure S3. **Additional Qualitative Results (SDD)**.