Supplementary Information: Smart Cellular Bricks for Decentralized Shape Classification and Damage Recovery

Rodrigo Moreno[1,*], Andres Faina[1,*], Shyam Sudhakaran[2], Kathryn Walker[1], Sebastian Risi[1,3,+]

[1]IT University Copenhagen, Rued Langgaards Vej 7, Copenhagen, 2300, Denmark

[2]Autodesk Research

[3]Sakana AI

[*]These authors contributed equally.

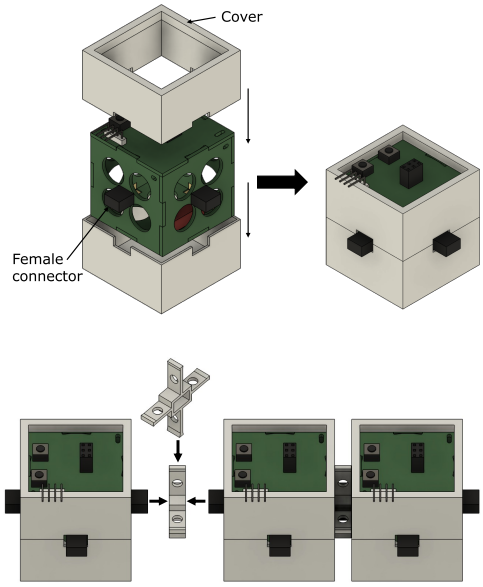
[+]email: sebastianrisi@sakana.ai

1 Training data example shapes

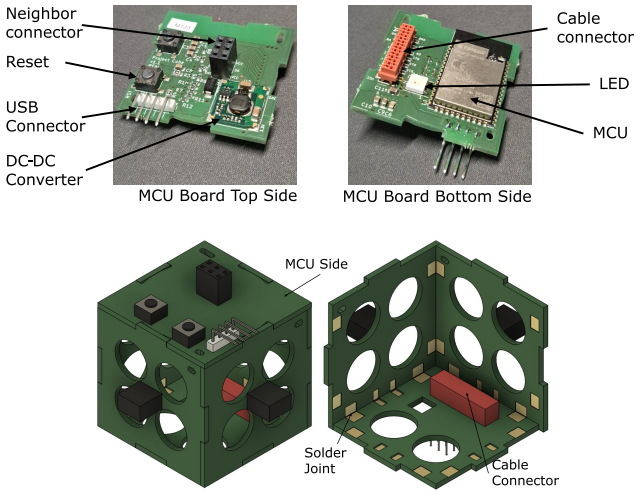
Supplementary Fig. 1: **Examples of shapes included in the training set**. The training set contains 487 instances of 7 different shapes (68 guitars, 114 boats, 6 houses, 125 tables, 10 cars, 61 chairs and 103 planes). The size of the instances varies from 16 to 562 cubes.

2 Cellular bricks mechanics and electronics

Details on the mechanics of the cellular bricks are shown in Supplementary Fig. 2, and details on the electronics in Supplementary Fig. 3.



Supplementary Fig. 2: **Mechanics**. (Top) The 5 electronics boards of the bottom of the bricks are soldered together and the top board carrying the MCU is assembled with cable ties. After that, a 3D printed white cover is assembled to diffuse the light. (Bottom) In large assemblies, a spacer is used in between bricks to distribute the load more evenly among the cube face.



Supplementary Fig. 3: **Electronics of the cellular bricks**. (Top Right) The bottom side of the microcontroller (MCU) board contains an ESP32-S2 WROVER Wifi module that manages all the processing and communication in the brick modules. Next to it lies a WS2812B RGB LED and a 14 way flat cable connector. (Top Left) The top side contains a diode protected DC-DC converter that powers the ESP32 module and the LED, and the outside connectors to USB and the brike's neighbors, as well as the reset button and bootloader mode button. (Bottom) Four FR4 boards containing a header or socket connector are used as lateral sides. A fifth board containing also a 14 way micro-match connector fits the bottom. Plated pads on the edges are soldered together to provide structural rigidity and power and communication connections.

3 Communication test and data collection

To ensure that all data connections between bricks are working correctly we prompt all bricks to send their unique identifier number through all their faces. To achieve this we implemented the sending of a special message in which all the bytes represents the same number when we press the boot mode button on the MCU board. The message is sent to the neighboring modules which in turn change their mode and propagate the message. In this way, we can prompt all modules to send their identifier by pressing a single button, without regard to the number of modules.

We use the MAC number from the WiFi module as unique identifier. The received information is collected in a computer by requesting it from an http server running in the microcontroller (using the ESPAsyncWebServer Arduino library), and a python script analyzes the connections and compares it to the intended shape, detecting missing connections and reconstructing the shape from the existing ones. This same server is used to toggle the different functional modes, from running the classification neural network to getting into a *failure* state. The mode information is stored in non volatile memory to ensure that all modules start in the correct mode after a reset. The current mode can only be changed while the bricks are connected to the Wifi network, which only happens after a classification test or when prompting the bricks to send their id. The Wifi peripheral of the microcontroller is deactivated during tests to prevent bad interrupt timings that could affect communication. During a classification test the internal cell state, the current guess, the current time in ms from when the bricks started and whether the brick is in a failure state are stored at each forward pass step. After the 60 steps the brick connects to a central server in the WiFi network using an http client (using the HTTPClient Arduino library) and posts all this information to a database.

4 Baseline comparison

In this paper, we address the problem of classifying 3D modular shapes represented as voxel assemblies. To the best of our knowledge, no prior method enables shape classification in a fully decentralized setting, where individual modules operate using only local communication.

To provide a baseline comparison to our NCA approach, we adapt the Weisfeiler–Lehman (WL) graph kernel framework to a distributed context. Here, each module computes local geometric features that are iteratively re-

fined through neighborhood-based message passing, effectively propagating structural information across the shape. These locally computed features collectively encode a global representation of the structure, which is subsequently used for supervised classification. Crucially, the entire feature extraction process is inherently local, allowing every module to independently reconstruct the same global feature vector without centralized coordination. This enables each module to run an identical classifier (e.g. a linear SVM) and reach a consistent prediction of the overall shape. While these experiments are conducted in simulation, where voxel assemblies are explicitly represented as graphs, the proposed method is directly compatible with our hardware platform and can be implemented as a fully distributed algorithm.

4.1 Voxel Graph Weisfeiler–Lehman baseline

Each voxel assembly is represented as an undirected graph $G = (V, E)$, where each occupied voxel corresponds to a node $v \in V$. Edges $(u, v) \in E$ are defined between voxels that are adjacent in the 6-neighborhood (axis-aligned connectivity). Two voxels are connected if their coordinates differ by one unit along a single axis. This representation captures the local structural connectivity of the shape. Each node is assigned an initial label encoding local geometric information, consisting of (1) its degree (number of neighbors) and (2) the number of neighbors along each spatial axis (x, y, z). Formally, the initial label of node v is defined as:

$$\ell_v^{(0)} = \mathtt{d}_{\deg(v)}\_\mathtt{x}_{n_x(v)}\_\mathtt{y}_{n_y(v)}\_\mathtt{z}_{n_z(v)}, \tag{1}$$

where $n_x(v)$, $n_y(v)$, and $n_z(v)$ denote the number of neighbors of v along each axis. This enriched labeling captures anisotropic local structure beyond simple node degree. We apply H iterations of the Weisfeiler–Lehman refinement procedure. At each iteration t, node labels are updated based on their current label and the multiset of labels of their neighbors:

$$\ell_v^{(t+1)} = \text{Compress}\left(\ell_v^{(t)}, \{\{\ell_u^{(t)} : u \in N(v)\}\}\right), \tag{2}$$

where $N(v)$ denotes the set of neighbors of node v, and $!\cdot!$ represents a multiset. In practice, neighbor labels are sorted to obtain a canonical representation, and concatenated with the current node label to form a signature. A global canonicalization step then assigns a unique new label to each distinct signature, ensuring that nodes with identical signatures receive the same label while different signatures are mapped to different labels. After

H iterations, node labels encode structural information from neighborhoods of radius up to H.

We construct a graph-level feature vector by aggregating node labels across all WL iterations. Specifically, we compute a histogram of node labels at each iteration $t = 0, \ldots, H$, and concatenate them:

$$\phi(G) = \bigcup_{t=0}^{H} \text{hist}\left(\{\ell_v^{(t)} : v \in V\}\right). \tag{3}$$

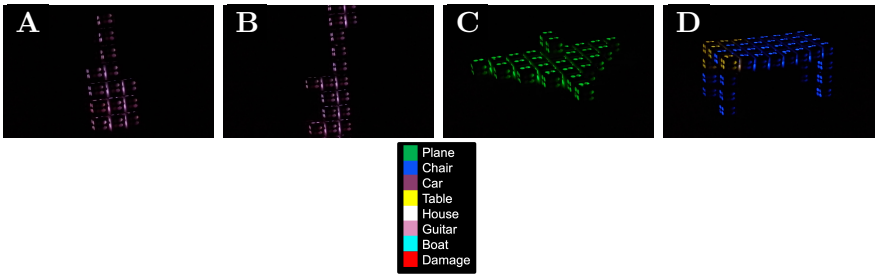
This representation corresponds to the Weisfeiler–Lehman subtree kernel feature map, capturing both local and increasingly global structural patterns. To obtain fixed-dimensional feature vectors, we construct a global vocabulary of labels observed in the training set. Each graph is then encoded as a sparse vector counting occurrences of each label in this vocabulary. This ensures that all graphs are embedded in a common feature space.

The resulting feature vectors are used to train a linear Support Vector Machine (SVM). We apply feature standardization (without centering) followed by a linear SVM with regularization parameter C, which controls the trade-off between margin maximization and classification error. In our experiments, we set $C = 1.0$. Given a new shape, its WL histogram features are computed using the same label vocabulary, and the trained classifier predicts its class label.

The proposed method combines geometric node descriptors with WL-based structural feature extraction. The iterative refinement process captures increasingly global shape information through local message passing, while the final classifier operates on a fixed-dimensional representation of the entire structure.

4.2 Classification performance

We evaluated the classification performance of the Voxel Graph Weisfeiler Lehman (WL) baseline on the dataset, achieving 100% classification accuracy, comparable to the 98.97% obtained by the NCA model. To assess generalization beyond the training distribution, we conducted additional experiments under out-of-distribution conditions. Specifically, we considered: (1) the same set of out-of-distribution shapes used to evaluate the NCA, (2) the same shapes subjected to identical damage patterns as in the NCA robustness experiments, and (3) four additional datasets generated by uniformly scaling the original shapes by factors of 0.5, 0.8, 1.5, and 2.0.



Supplementary Fig. 4: **Selection of smaller shapes after 60 NCA classification steps**. (**A**, **B**) Two small guitars that are correctly classified. (**C**) A small plane that is correctly classified. (**D**) A small table that is recognized as a chair.

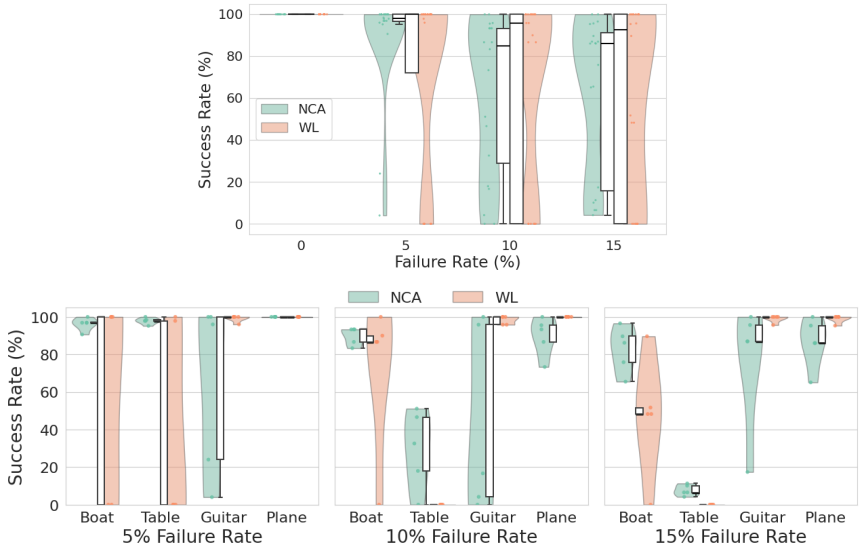
Regarding the out of distribution shapes, we tested 6 small versions of two guitars (Supplementary Fig. 4A and B), two planes (Fig. 3F and Supplementary Fig. 4C) and two tables (Fig. 3G and Supplementary Fig. 4D), a damaged table (Fig. 3D) and modified boat (Fig. 3E). All NCA experiments were performed in hardware. The results show that both approaches correctly classified all shapes except the small tables (Supplementary Table 1). WL classifies both small tables as planes and the NCA classifies the majority of cells as chairs (Fig. 3G and Supplementary Fig. 4D).

Supplementary Table 1: Classification accuracy (%) on out-of-distribution shapes under different conditions.

Shape	**Condition**	**WL**	**NCA**
Guitar 9	Small	100.0	100.0
Guitar 22	Small	100.0	85.0
Plane 213	Small	100.0	100.0
Plane 261	Small	100.0	100.0
Table 372	Small	0.0	0.0
Table 393	Small	0.0	6.8
Table 372	Damaged	100.0	100.0
Boat 144	Damaged	100.0	100.0

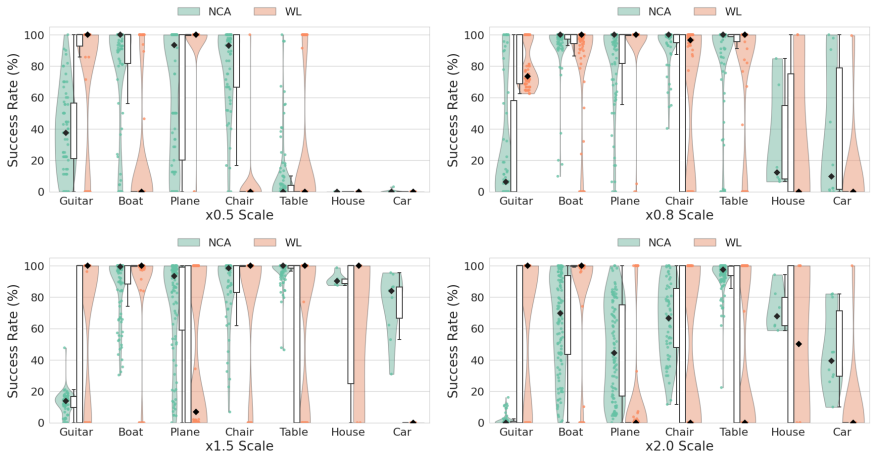
Regarding the robustness to faulty cells, we reconstructed all the shapes that we tested in the NCA experiment and classified them with the WL approach. The results indicate that the performance of both techniques

is comparable and that the success rate decreases with the percentage of faulty cells (Supplementary Fig. 5). The WL approach seems more robust classifying guitars and planes and the NCA seems more robust classifying boats and tables.



Supplementary Fig. 5: **Accuracy for shapes with failure rates**. Violin plots show the distribution density of the results for each category. Box plots indicate the median (center line), interquartile range (box), and whiskers extending to 1.5× the interquartile range. Individual points represent all experimental observations. (Top) The performance between both algorithms is similar and decreases with the percentage of damaged cells. (Bottom) The extent of performance degradation depends strongly on the shape.

Finally, we evaluated the performance of both approaches on scaled versions of the dataset. In this setting, both methods were tested in simulation. As shown in Supplementary Fig. 6, the two approaches achieve comparable performance, with an average success rate of 60.3% ± 48.0 for the WL baseline and 66.7% ± 39.0 for the NCA across all scales. For both methods, a significant degradation in performance is observed at extreme scaling factors (×0.5 and ×2.0), particularly for shapes such as houses and boats, likely due to their limited representation in the training dataset. The WL baseline further exhibits a bimodal performance distribution, which can be attributed to the global aggregation of labels prior to classification. As a result, predictions tend to be either highly accurate or largely incorrect, with



Supplementary Fig. 6: **Classification accuracy for scaled shapes**. Both approaches perform similarly but significant performance degradation is observed in the extremes zooms levels (x0.5 and x2.0). Diamond markers indicate the median value for each distribution.

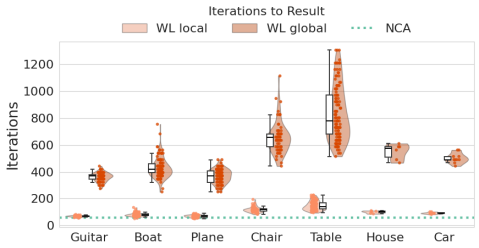
intermediate values occurring when the shape becomes disconnected.

4.3 Convergence speed performance

We analyze the time required to obtain a classification result in a distributed setting (see Supplementary Fig. 7). As a reference, the NCA requires a fixed number of iterations (60 in our experiments), represented as a horizontal dotted line in the figure. For the WL-based approach, we consider two implementations that differ in how node labels are updated at each iteration.

In the first variant, nodes exchange their current labels with their neighbors and construct a local signature by combining their own label with the multiset of neighbor labels. These signatures are then transmitted to a central coordinator, which performs a global canonicalization step by assigning a unique label to each distinct signature. The updated labels are subsequently broadcast back to all nodes, and the next iteration begins. This approach ensures exact WL label consistency, but incurs significant communication overhead due to repeated global aggregation and broadcast at every iteration.

In the second variant, nodes compute the new labels locally using a deterministic hash function. Each node constructs its signature from its own label and its neighbors' labels, and directly applies a hash function to ob-



Supplementary Fig. 7: **Convergence speed comparison beween the WL and NCA approaches**. Violin plots show the distribution density of the results for each category. Box plots indicate the median (center line), interquartile range (box), and whiskers extending to 1.5× the interquartile range. Individual points represent all experimental observations. The NCA baseline requires a fixed number of local iterations (60 in our experiments), shown as a horizontal dashed line. The WL approach with global label canonicalization incurs a communication cost proportional to the graph diameter D at every iteration, due to repeated global aggregation and broadcast. In contrast, WL with local hash-based labeling avoids per-iteration global communication, requiring it only during the initial structure discovery and final aggregation phases. Although the hash-based variant significantly reduces latency, both WL methods remain dominated by these global communication phases, which scale with D.

tain the updated label. This removes the need for global coordination during each iteration, reducing communication to local exchanges only. While this approach introduces a theoretical risk of hash collisions, it significantly improves efficiency in practice.

In both cases, the WL algorithm is executed for H iterations (with $H = 10$ in our experiments). After the final iteration, node-level labels are aggregated to form a global feature representation, which is used for classification.

We propose a simple analytical model to estimate the time-to-result of the distributed system. The model assumes that communication occurs over local connections and that the cost of global coordination depends on the graph diameter. Let D denote the diameter of the shape, defined as the maximum shortest-path distance between any pair of nodes. The total execution time is decomposed into three phases:

1. **Structure discovery and coordinator election.** A spanning tree is constructed and a coordinator is selected using local communication,

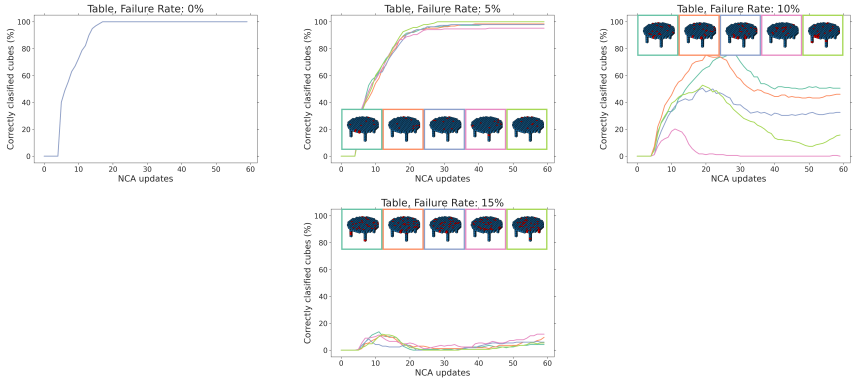
requiring approximately $2D$ communication rounds. This corresponds to a forward propagation of information across the structure and a subsequent convergecast phase, as commonly employed in distributed modular robotic systems.

2. **WL iterations.**
   - *Local hash-based labeling:* each iteration requires only local communication, resulting in a total cost of H rounds.
   - *Global label canonicalization:* each iteration requires both local communication and global aggregation/broadcast, resulting in a cost of $H(1 + 2D)$ rounds.
3. **Feature aggregation and dissemination.** After the final iteration, node-level features are aggregated at the coordinator and the resulting classification (or feature vector) is broadcast back to all nodes, requiring an additional $2D$ rounds.

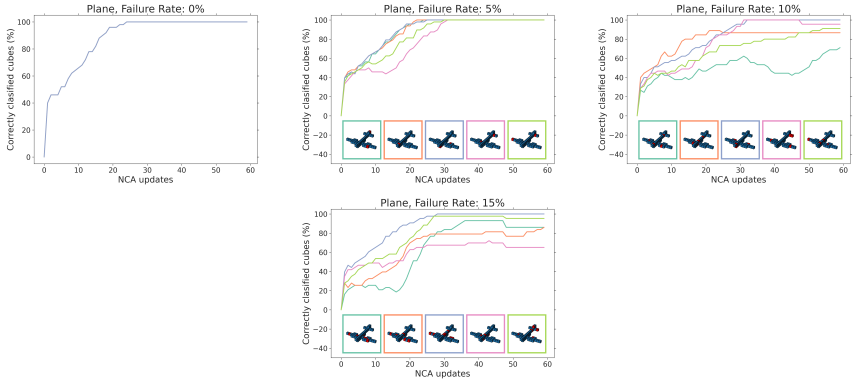
As shown in Supplementary Fig. 7, the NCA baseline achieves significantly lower time-to-result due to its purely local update rule and absence of global coordination. In contrast, the WL approach with global label canonicalization is dominated by repeated global communication, resulting in substantially higher latency. The local hash-based variant reduces this cost considerably, but still incurs overhead from the initial structure discovery and final aggregation phases. This highlights the importance of minimizing global communication in distributed modular systems.

5 Detailed results on robustness to faulty cells

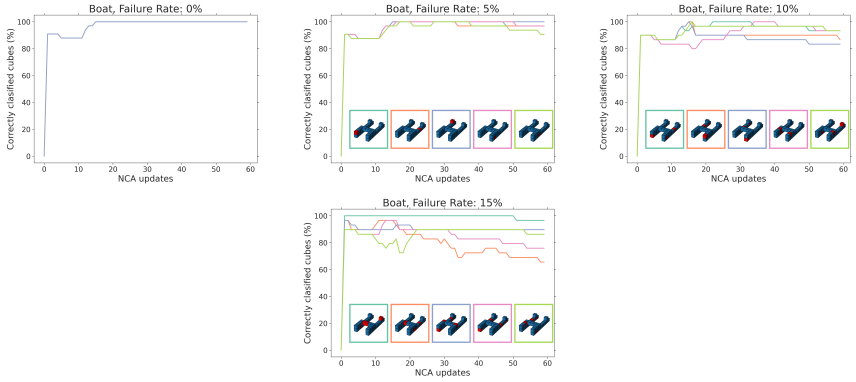
Supplementary Figs. 8, 9, 10, and 11 show more detailed hardware results for a variety of shapes under varying levels of faulty cells.



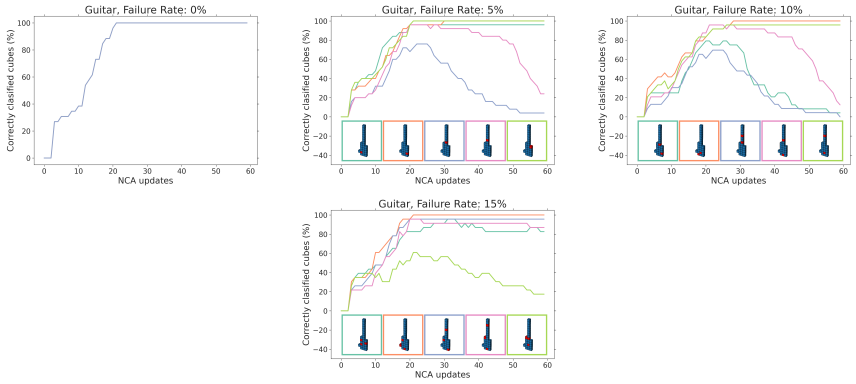
Supplementary Fig. 8: **Classification experiments for the table in hardware**. The four graphs show the percentage of correctly classified bricks for failure rates of 0, 5, 10 and 15%. The damaged modules for each experiment are indicated with small icons inside the graphs.



Supplementary Fig. 9: **Classification experiments for the plane in hardware**. The four graphs show the percentage of correctly classified bricks for failure rates of 0, 5, 10 and 15%. The damaged modules for each experiment are indicated with small icons inside the graphs.



Supplementary Fig. 10: **Classification experiments for the boat in hardware**. The four graphs show the percentage of correctly classified bricks for failure rates of 0, 5, 10 and 15%. The damaged modules for each experiment are indicated with small icons inside the graphs.



Supplementary Fig. 11: **Classification experiments for the guitar in hardware**. The four graphs show the percentage of correctly classified bricks for failure rates of 0, 5, 10 and 15%. The damaged modules for each experiment are indicated with small icons inside the graphs.