Harnessing Simulated Datasets with Graphs

Thesis Dissertation Proposal

Columbia University

Henrique Teles Maia

December 31, 2021

Abstract

Physically accurate simulations enable the generation of limitless data from within meticulously crafted environments. Digital sandboxes allow for collecting observations at-will and in arbitrary scenarios. This is particularly useful given the ubiquity of data-driven techniques across engineering disciplines. These are systems which inform behavior from aggregating over available samples. Paired together with statistical methods, the ability to synthesize endless data promises to make approaches that benefit from datasets all the more robust and desirable. However, if not careful, pipelines that naively aim to measure virtual scenarios can easily be overwhelmed by trying to sample an infinite set of available configurations. Variations observed across multiple dimensions can quickly lead to a daunting expansion of states, which must be processed and solved. Therefore we propose to wield graphs in order to instill structure over captured data, and curb the growth of variables. The graphs we introduce serve to enforce consistency, localize operators, and crucially factor out any combinatorial explosion of states. We demonstrate the effectiveness of this methodology in three distinct areas, each offering their own challenges and practical constraints. Namely, we observe state-of-the-art contributions in design for additive manufacturing, side-channel security threats, and large-scale physics based contact simulations, that are achieved solely by harnessing simulated datasets with graph algorithms.

Contents

Chapter 1

Introduction

The objective of the research described in this dissertation is to channel the efforts required to leverage large handcrafted datasets by structuring the observations as graphs. Most data-driven techniques place the burden of sorting and sifting through features in their data to downstream optimization techniques. However, virtual measurements make it easy to capture an abundance of labeled entries in diverse and arbitrary states. Adequately sampling these configurations leads to an exponential and combinatorial growth of the dataset that cause difficulties to most numerical methods. Is it therefore necessary to first shape and funnel features drawn from large quantities of complex, high dimensional, and interdependent variables.

We find graphs are generally suited to assembling dependencies that are easier to operate over. Graphs map domain expertise into assumptions, constraints, and connections that can be generalized over the data. This added consistency complements the dataset by assisting with the combinatorics of the observed samples. Therefore, through careful structuring of the data into graph-like structures, we can safely expose our algorithms to varied conditions with an understanding of how the system will perform and scale.

The benefit of such an approach is the ability to tackle real and physical constraints via an unconstrained dataset of hand-crafted and curated observations. Systems that utilize graph treatment over instances of the data are flexible. Such methods can adapt to new situations, without additional computational burden, by simply adjusting the data used to drive the pipeline as new data becomes available. We particularly focus on data collected from simulated and controlled environments, since these provide for physically accurate and realistic models of behavior. We present a methodology that outlines how to cast synthetic datasets into a structure whose processing guarantees simpler frameworks and ultimately more robust applications.

1.1 Methodology

The goal of many researchers to achieve and model realistic behavior. We propose a methodology for collecting and utilizing large sets of simulated data to inform realistic models of behavior in conjunction with graph algorithms. Simulated and synthetic examples are not limited like their real world annotated counterparts. Digital observations are collected at low cost, can be automated, and permit studying of arbitrary scenarios and interactions. Their exploratory discretion is not without fault however, as this added freedom also adds both complexity and degrees of freedom to the measurements taken. In turn, generalized patterns must be formulated across the dataset to reduce this combinatorial expansion of variables. Graphs restrain the space of features considered by focusing computational operations on nodes and edges. Categorizing data into these primitives also simplifies the relationships between values by explicitly laying out dependencies. Therefore, we believe that graphs are specifically positioned to assist with data driven approaches that have exploded in popularity over the past decade.

Our methodology is as follows:

1. Generate a specialized dataset from synthetic data to assist an objective
2. Identify combinatorial challenges and inter-sample patterns to factor
3. Overlay a graph to instill structure and manage the spectrum of states
4. Leverage graph algorithms to operate on the data via this distilled lens

Contributions

We investigate challenges receptive to digitally acquired datasets from applications in diverse problem areas. We validate this methodology as it applies to computer vision tasks on real world 3D printed objects, side channel extraction of neural network topologies from active GPUs, and lastly efficient strand contact simulation. We show not only that this approach is broadly applicable, but that systems that partner data driven models with graph algorithms outperform existing methods in terms of robustness and quality guarantees.

1.2 Roadmap

The product of this dissertation will be a study of the benefits to marrying graph structures with simulated data collection in modeling various behaviors. Our proposed assessment consists of three studies:

1. **LayerCodes** (Ch. 3) We observe how simulated proxies can assist with the design of physical tags that embedd information in additive manufactured geometries. The resulting approach relies on graph structures to produce a robust tag given the complex interplay of unconstrained geometries present in the dataset.

2. **Neural Snooping** (Ch. 4) Diverse network architectures are distilled into graphs for extraction and reconstruction through electro-magnetic side channel attacks. Through carefully curated synthetic datasets we are able to recover large and deep black-box designs to great fidelity.

3. **Data-Driven Hair Contact** (Ch. 5) We propose to apply our methodology against the inefficiencies of contact resolution in large scale elastic-rod interactions by training Graph Neural Networks on simulation data.

Taken together, these evaluations survey the range of applications that stand to benefit by combining powerful graph techniques with limitless synthetic data points.

Chapter 2

Related Work

Here we overview recent advances in data-driven techniques broken into several general areas:

1. **Sim-2-Real & Real-2-Sim**: Bridging real world measurements and applications with digital models and representations.

2. **Data-Driven Techniques**: Leveraging statistical analysis over collected databases to inform heuristics and algorithms.

3. **Graph Algorithms**: The involvement of graphs, trees, and more contemporary techniques to instill structure over data within algorithms.

Sim-2-Real & Real-2-Sim The task of Sim-to-Real, and conversely Real-to-Sim, transfer is one shared by many application areas. Simulations provide controlled environments for gathering observations and experimentation at low cost. However, their validity is highly dependent on an accurate representation of the real world.

Simulated models are often intended to be used in the real world. Applications in robotics design practice in digital sandboxes before entering reality [23, 40, 50]. Similarly, techniques in computer vision are often tailored to interpret real images and align them with software interactions [11, 22, 48]. The gap between simulation and reality must also be faced when preparing security applications, as software and hardware considerations must be taken into account to contend with digital and physical attacks [1, 20, 34].

On the other hand, it is often fruitful to guide the development of a digital technique with real world measurements. Principled values facilitate

parameter tuning in the case of physics-based simulations, which combine mechanical and physical equations to produce dynamic behavior [5, 13, 24]. Careful understanding and measurement of the real world can also inform design. Such is the case most prominently when manufacturing features that depend on physical interactions, as in the fields of physical information embedding [27, 28, 35] and digital design for fabrication [44, 46, 45].

Data-Driven Techniques Curated datasets provide alternatives when precise behavioral models are unavailable or expensive. Often these measurements provide an input-output pair collected by labeling real world data, as is often the case with image based datasets [9, 10]. These datasets feature practical scenarios in the wild, but can often times be laborious or subjective to label. Data-driven techniques can also benefit from synthetic collections, such as those comprised of 3D objects or paths [3, 15, 53], as is often of interest to those in robotics. Regardless of their form, these datasets describe aggregate behavior that can be used to statistically dictate algorithms.

Data-driven models approximate complexity in a variety of ways. Model-reduction techniques couple datasets with simplified or scaled behavioral models. These include methods that make simplifying assumptions about dynamic behavior [12, 13], or project degrees of freedom into a lower dimensional—and thereby easier to solve—space [14, 51]. Machine learning approaches pair neural networks with data to great effect. By linking results to data features through gradient descent and back-propagation, applications in vision [19], security [52], natural language processing [49], and robotics [25], all stand to benefit from larger datasets when possible, and have been shown to work with both synthetic and real data.

Graph Algorithms Mathematical graph theory extends into computer science by way of graph data types and traversal algorithms. These approaches define rules over collections of nodes and edges, attributing each with specific properties and relationships to one another [18, 37]. This gives structure abstractly to data formulated in such primitives, e.g. limiting data in a graph node to only interact with incident edges or neighboring nodes.

Recently these approaches have been combined with machine learning techniques to directly benefit from datasets. The result is the introduction of Graph (Neural) Networks who host embeddings on nodes and edges of a graph that can influence one another through neural-graph hybrid operations [2, 38, 42]; and profit from both data-driven and graph techniques.

Chapter 3

LayerCodes

With the ongoing advance of personal and customized fabrication techniques, the capability to embed information in physical objects has become both more crucial and challenging. Traditionally optical barcodes serve as the indispensable link that bridges physical artifacts to modern digital systems. However, barcodes rely on constrained industrial designs that mandate flat and smooth geometries. In this work, we rethink barcodes in the context of additive manufacturing, popularly known as 3D printing. 3D printing offers a quick way of making customized, complex shaped objects. Unlike a mass-produced product which by design has a reserved flat surface region to host barcodes, 3D printed shapes are often complex and curved: thin features, slender threads, and holes are not uncommon. As a result, the traditional barcodes cannot be placed or printed on such objects.

We apply our methodology to improve the optical tagging of custom 3D shapes. We motivate our design by generating a dataset of 4,835 printable shapes, along with image renderings of our objects from various views, that allows us to experiment and preview challenges specific to customized geometries. We study this dataset to identify local invariants common to all shapes, regardless of holes or intricate features. These local properties are extracted from renderings of tagged objects and used to construct a graph, which abstracts out spatial complexities, allowing all custom shapes to be treated uniformly. Traversing this graph we achieve a robust in-plain-sight algorithm that enables the 3D printing layers to carry information without altering the object's geometry. We name our tagging scheme *LayerCode*, and demonstrate how a carefully designed pattern may be directly embedded in objects as a deliberate byproduct of the 3D printing process. We show that LayerCodes work across various types of 3D printers, and succeed on complex, nontrivial shapes, on which previous tagging mechanisms all fail.

Objective LayerCodes aims to bring the concept of optical barcodes into 3D printed objects, especially those with curved shapes and fine structures. Our key idea is inspired by a structural resemblance between optical barcodes and 3D printed objects: essential in a barcode are its two-tone bars arranged in parallel; universal in all 3D printed objects are the printing layers introduced in a parallel fashion. In fact, virtually all additive manufacturing uses a layer-by-layer printing process [32, 39]. Thus, if we could interleave two categories of layers in a 3D printing process—be it through color, texture or otherwise—we would be able to embed a tag everywhere in a 3D printed object.

Materializing this idea faces major algorithmic and combinatorial challenges. Due to an object's complex shape, its layering structure may appear curved, disconnected, or shadowed when captured by a camera. It is difficult to anticipate how these effects will interplay under perspective projection without first deploying any proposed encodings across a variety of objects. Moreover, unconstrained printable shape geometries, such as those allotted by customized fabrication, readily combine in a diversity and frequency of patterns that can easily overwhelm naive decoding approaches that fail to generalize. We therefore seek a robust encoding and decoding algorithm that embeds information in printing layers and later retrieves this information from the images of a conventional camera.

3.1 Method

We address these challenges by introducing a new coding algorithm and verifying it through a large dataset of simulated examples. Unlike the standard barcode that maps every bit to a fixed bar thickness, we encode individual bits based on the local change of layer thickness. We will show that such local changes are invariant under different surface orientations and curvatures. At decoding time, we exploit a key observation that each layer spans the entire cross-section of the object. This suggests that there exist many image-plane paths along which we can decode. The rich set of decoding paths is advantageous, enabling us to sidestep shadows, highlights, and uncertain image regions. Together these steps allow us to tag robustly, as supported through both simulated and real-world LayerCoded geometries.

Encoding

The input to our encoding algorithm is a 3D shape, the tag information represented as a bit string, as well as the printing direction with respect to the printed object (i.e., the direction along which 3D printing layers will be grown). Unlike other tagging methods, there is no restriction on the 3D printed shape. We leave the flexibility of choosing a printing direction to the user, because the printing direction may depend on the specific shape, printing software, support materials, and perhaps semantic or subjective preferences. The output of the encoding algorithm is a series of slices along the printing direction to specify the thickness of each coding layer.

Our key insight comes from noticing the fact that if the coding layers are thin (relative to the inverse of the surface curvature along the printing direction), the thickness *ratio* of two consecutive layers measured in a local region of the image plane is *invariant*. This is because in a small local region, two nearby coding layers share approximately the same surface tangent plane, and the projection from the tangent plane to the image plane follows an affine transformation which preserves the layer thickness ratio.

Using local thickness ratios also favors the decoding step. It allows us to sample the thickness ratio of two layers at many local regions on a captured image, and collectively estimate a thickness ratio that is robust against imaging noise and artifacts.

Coding scheme We propose the following scheme to encode every bit in a bitstring. A bit "**1**" is encoded if the thickness ratio of two consecutive layers is either $1/M$ or M, where M is a constant larger than 1 that we will discuss shortly, and a bit "**0**" is represented by a unitary thickness ratio (i.e., the same thickness). The representation of a bit string always starts from a layer with a baseline thickness h. The next layer thickness a_{n+1} is either h or Mh according to the current bit b_{n+1} and the previous layer thickness a_n, namely,

$$\mathrm{a}_{n+1} = \begin{cases} a_n & \text{if } b_{n+1} = 0, \\ Mh & \text{if } b_{n+1} = 1 \text{ and } a_n = h, \\ h & \text{if } b_{n+1} = 1 \text{ and } a_n = Mh. \end{cases} \tag{3.1}$$

At decoding time, we recover the bit string sequentially, using the inverse map

$$\mathrm{b}_{n+1} = \begin{cases} 1 & \text{if} \quad \log a_n - \log a_{n+1} = \pm \log M, \\ 0 & \text{if} \quad \log a_n - \log a_{n+1} = 0. \end{cases} \tag{3.2}$$

In practice, the value of $\log a_n - \log a_{n+1}$ will never be precisely $\pm \log M$ or 0 due to the image estimation errors. But a nice property of this coding scheme is that the estimated values of $\log a_n - \log a_{n+1}$, when viewed as a random variable, will form three distribution modes symmetrically centered at $\pm \log M$ and 0. We will later return to this property for robust decoding. Figure 3.1 illustrates this scheme for $M = 2$.

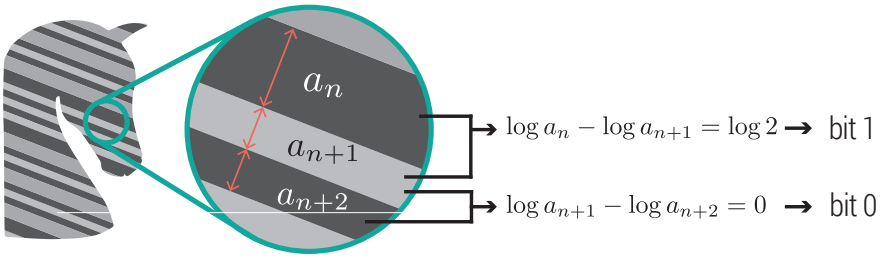

Figure 3.1: **Encoding scheme.** Pairs of layers encode a single bit. A bitwise 0 or 1 can be determined by computing the ratio of adjacent layer thicknesses.

For further details regarding choice of M, guarantees surrounding information capacity, and encoding bitstring direction, repetition, and error correction, please refer to the full paper [35].

Decoding

We propose a *graph-based* algorithm in order to robustly tackle the challenges associated with decoding the proposed scheme from image pixels of complex geometries. This is achieved by first constructing a graph to represent the potentially fragmented layer structure. We treat each coding layer region, which may not include an entire layer, as a graph node. Two nodes are connected if they are from different but neighboring layers.

Graph Construction Through a flood-fill process, we identify individual pixel regions where all pixels share a category as a preprocessing step. Each

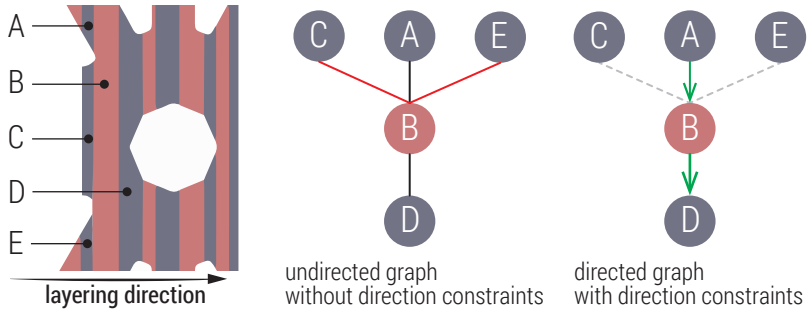

Figure 3.2: **Graph construction and traversal.** (left) Pixel regions are identified (A-E) through flood filling. (middle) Nodes are connected to form a graph if their regions are adjacent to each other. Since layers are added along the printing direction, it makes no sense to traverse backwards along a direction whilst decoding. Thus, A→B→C would not produce a valid bit string, while A→B→D is reasonable (right).

region is represented as a graph node, and two nodes are connected if their regions are adjacent to each other (3.2-a,b).

Next, we associate every graph edge e with two quantities, a 2D vector $\boldsymbol{v}$ in image space and a binary label r. Consider an edge e that connects nodes A and B. At each boundary pixel, we estimate a boundary *normal* direction as the direction along which we can enter into a different region by moving the shortest distance. $\boldsymbol{v}$ is then defined as the average normal direction over all boundary pixels between region A and region B. When computing the average, we use the normal direction $\boldsymbol{n}_p$ for pixel p in region A. Thus, the average direction $\boldsymbol{v}$ is in fact associated to the *directed edge* from A to B, and for clarity we denote it as $\boldsymbol{v}_{A\to B}$.

The binary label r is associated to the undirected edge, and is denoted as $r_{A\leftrightarrow B}$ for clarity. We compute $r_{A\leftrightarrow B}$ as follows. First, from each boundary pixel p between A and B, we estimate the layer thickness $h_A(p)$ of the region A by first finding the shortest image-plane vector $\boldsymbol{d}_{\mathrm{m}}$ between p and another region that is not A or B but connected to A. $h_A(p)$ is then set to be the length of $\boldsymbol{d}_{\mathrm{m}}$ projected on the normal direction $\boldsymbol{n}_p$ Symmetrically, from p, we also estimate the layer thickness $h_B(p)$ of B using a similar step. Then, pixel p contributes a vote for $r_{A\leftrightarrow B}$. It votes for label "0" if $|\log h_A(p) - \log h_B(p)| < \frac{1}{2}\log M$ (i.e., closer to 0), indicating the second case in eq. 3.2 and suggesting a bit "0" encoded between A and B. On the other hand, if $|\log h_A(p) - \log h_B(p)| \geq \frac{1}{2}\log M$, it votes for label "1", suggesting the first case in eq. 3.2 and hence a bit "1". The final label $r_{A\leftrightarrow B}$ is taken as the majority vote over all boundary pixels.

At first glance, assigning the label $r_{A\leftrightarrow B}$ requires a prior knowledge of M, which is not known from the image. Fortunately, our coding scheme presented in 3.1 enables an easy and robust way of estimating $\log M$. In the above process, we collect all $|\log h_A(p) - \log h_B(p)|$ values for all boundary pixels on the image. From 3.2, we know that these values are expected to be either $\log M$ or 0, although we do not know what M is. If we think of each $|\log h_A(p) - \log h_B(p)|$ value as a random variable, these random variables must be generated through a mixture of two Gaussians (in 1D): one is centered at 0, and another center (i.e., $\log M$) is unknown but can be estimated using the maximum likelihood estimation [36].

Decoding through Graph Traversal We now decode the bit string by traversing the graph node-by-node in a depth-first manner. Because the object is always 3D printed in a layer-by-layer fashion, we must avoid looping back to earlier layers during the traversal. To this end, the direction vector, $\boldsymbol{v}$, associated to each edge is helpful. As illustrated in 3.2-c, consider a traversal that reaches a node B from a node A. In the DFS, we visit the next node D, only when the moving direction from A to B is approximately consistent with the moving direction from B to D. In other words, we require $\boldsymbol{v}_{A\to B} \cdot \boldsymbol{v}_{B\to D} \geq \Delta$ ($\Delta = 0.35$ in all our examples).

This graph traversal process generates many paths and thus many bit strings. Some of them might be erroneous due to image noise. But collectively, they are robust. Therefore, we finalize the bit string by taking a bit-wise majority vote over all decoded bit strings.

For further remarks that improve the robustness and efficiency of decoding we defer the reader to the original work [35].

3.2 Evaluation

To validate the performance of our coding algorithm thoroughly, we must test our algorithm on a large diversity of shapes. In large part thanks to the ease and faithfulness generated by a photorealistic renderer, we are able to sidestep practical concerns required to develop such a validating dataset through the use of synthetic examples. A glimpse of the tested shapes is shown in Figure A.1. This evaluation over such a virtual dataset is justified by several considerations:

i. *Cost and time.* In terms of both cost and time, it is unaffordable to 3D print all the shapes in the dataset. 3D printing of a single object is usually an hour-long process, barring failures. Virtual rendering of 3D printed objects, on the other hand, can be finished in a short time, and the resulting images are photorealistic.

ii. *Feasibility.* For many complex shapes that we use in this evaluation, it is hard, if not impossible, to fabricate them via current commodity 3D printers. But 3D printing technology is constantly and rapidly improving. Therefore, it is desirable to test our algorithm on those complex shapes to prepare for the future.

iii. *Thoroughness.* In a virtual environment, we can test our algorithm using a large number of objects viewed from many camera angles. Thoroughly testing over all these variances provides us statistical insights which in turn guide our use of LayerCode tags in practice. This thoroughness is made possible only through simulated experiments.

Dataset

We tested our algorithm over a set of shape meshes from the Thingi10k dataset [53]. The testing shapes are selected through the following "printability" criteria: 1) They must be watertight 2-manifolds (i.e., no self-intersections), and 2) consist of a single connected component. 3) They should also have consistent surface normals without degenerate faces.

Following these criteria, we obtain **4,835** meshes. Each of these meshes is processed to embed a LayerCode tag indicating the mesh's database ID. When we encode the tag (using the procedure discussed previously), the printing direction is chosen to be the longest dimension of the mesh, and the baseline layer thickness h is set to repeat the tag three times. The output of the encoding step is a shape with two sets of coding layers ready for rendering. Each type of layer is assigned a different material color (i.e., red and blue). We then use the physics-based renderer Mitsuba [21] to generate a photorealistic image from a given camera angle.

To understand how the view angles affect the decoding, we uniformly sample 30 viewing directions on a sphere co-centered with the object. All the view directions are guaranteed far from the printing direction, since looking along the printing direction unlikely displays the entire barcode. Figure A.1 shows 18 representative shapes and the rendered images from multiple view

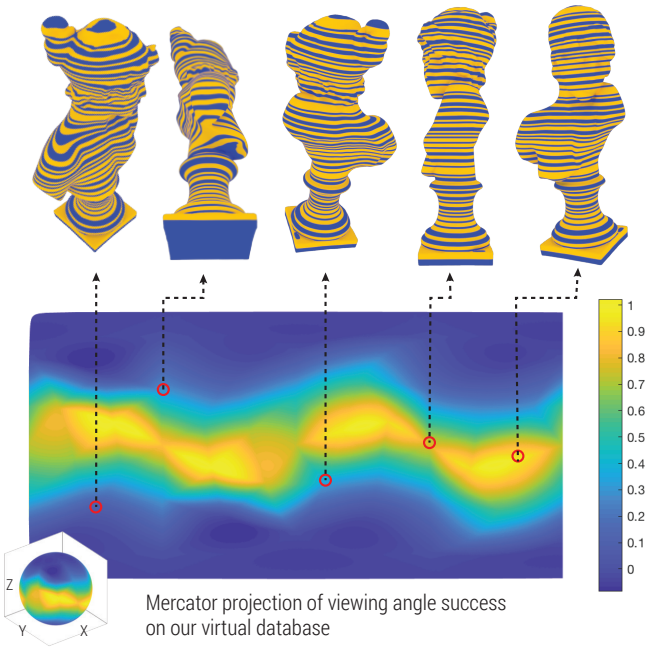

Figure 3.3: **The decoding success rate** of each view direction is color-mapped to a sphere, whose equatorial plane is perpendicular to the printing direction. This mapping is unrolled in the Mercator projection, with representative views shown (on top) for a few points of interest.

angles. The image from each view is the only input to the decoding algorithm, and thus each view is decoded independently.

3.3 Experiments & Results

Camera angle dependency Because of the surface curvature and local occlusions, from certain camera angles the coding layers are better seen. A natural question is what camera angles are more suitable for decoding the tag. Figure 3.3 reports our experiment results, suggesting that view directions just north or south of the equator appear statistically the most promising for decoding tags.

Figure 3.4 shows that some shapes can accommodate a wider range of view angles than others for successful decoding. For example, one shape is readable from all 30 views, whereas 44 other shapes are not decodable at all (which account for only 0.9% of the shapes in the dataset). On average, for any given shape, its tag is readable from 51% of the viewing directions

sampled. Overall, 78.0% of the shapes can be decoded in 10 views, 49.5% can be decoded in 15 directions, and 21.7% can succeed in 20 directions.

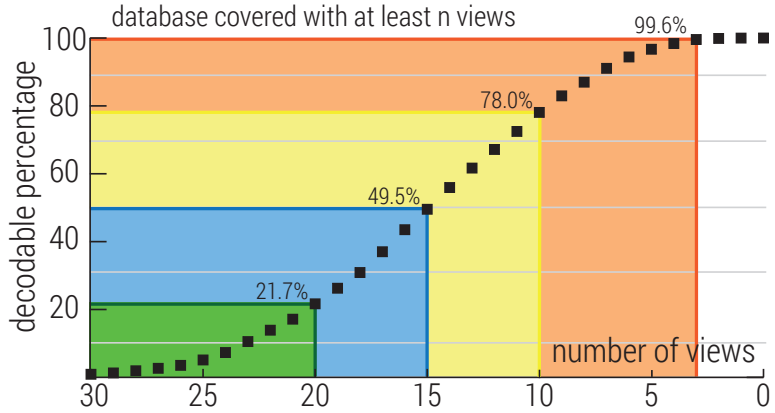

Figure 3.4: We plot the distribution of all 4,835 tested shapes with respect to the number of view angles from which they can be decoded successfully. 99.6% of the shapes can be decoded from at least one sampled view direction.

Lower bound of h. A smaller h allows the object to host more copies of the tag. But if h is too small, the coding layers will become hardly discernible on the image. In an experiment, we progressively reduce h and encode only a single copy of an ID in the object. In this process, we keep the camera angle and image resolution unchanged, and check at what h value the decoding would fail. Not surprisingly, the lower bound of h depends on the object shape. Figure 3.5 reports the results.

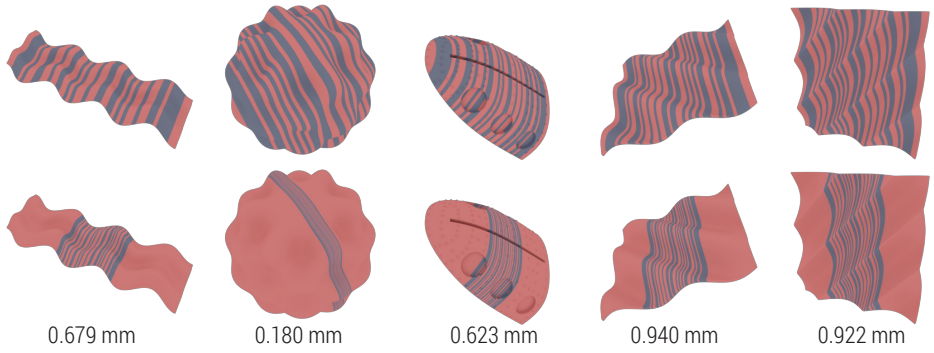

Figure 3.5: **Lower bound of h.** Here we show the smallest baseline layer thickness h still readable under different views for shapes normalized to 10cm in length along the printing direction.

Chapter 4

Neural Snooping

The graphics processing unit (GPU) is a favored vehicle for executing a neural network. GPUs allow difficult and sizable jobs to be treated faster, and have been used extensively in state of the art machine learning pipelines across both academic and commercial settings. The use of GPUs is motivated by the need to tune neural network applications meticulously for each task, since designs that can robustly resolve queries end up in high demand. As the commercial value of accurate and performant machine learning models increases, so too does the demand to protect neural architectures as confidential investments. This requires a careful look at the GPU.

We apply our methodology to explore the vulnerability of neural networks deployed as black boxes across accelerated hardware through electromagnetic side channels. We simulate a large number of queries across diverse neural architectures and gather side channel measurements with a cheap induction sensor. Examining the collected signals, we discover local patterns associated with individual computational steps, that taken together form the layers and blocks that define neural networks. We distill graphs from these observations that match the topology of network models and extraction of layer-specific attributes. By optimizing over these graphs and deducing crucial parameters, we exploit a robust side-channel signal that is able to generalize across both networks and hardware. We demonstrate the potential accuracy of this side channel attack in recovering the details required for a broad range of network architectures and attack scenarios.

Objective We wish to know the extent to which networks produce an identifiable magnetic signature, from which layer topology, width, function type, and sequence order can be inferred. Our primary focus centers on using an electromagnetic side channel to reverse engineer neural architectures and

their defining layer parameters. The networks in question may be of arbitrary size and depth, and involve combinations of fully connected, convolutional, and recurrent layers, along with a medley of interspersed activation, normalization, and pooling layers. Together these layers span the basic components used to assemble most state of the art networks and account for models used across a variety of machine learning applications [19, 52, 49, 25, 47, 26].

4.1 Method

We examine the magnetic flux emanating from a graphics processing unit's power cable, as acquired by a cheap $3 induction sensor, and find that this signal betrays the detailed topology and hyperparameters of a black-box neural network model.

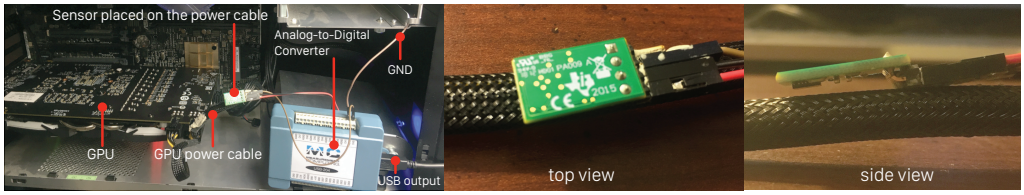

Figure 4.1: **Sensing setup.** Placement of the magnetic induction sensor on the power cord works regardless of the GPU model, providing a common weak-spot to enable current-based magnetic side-channel attacks.

To reconstruct the black-box network's structure from the acquired signal, we propose a two-step approach. First, we estimate the network topology, such as the number and types of layers, and types of activation functions, using a suitably trained neural network classifier. Then, for each layer, we estimate its hyperparameters using another set of deep neural network (DNN) models. The individually estimated hyperparameters are then jointly optimized by solving an integer programming problem to enforce consistency between the layers. We demonstrate the potential accuracy of this side-channel attack in recovering the details for a wide range of networks, including large, deep networks such as ResNet101 [19]. We further apply this recovery approach to demonstrate black-box adversarial transfer attacks.

Topology Recovery

Since neural models function to process input data into an output, they can be unrolled into a directed acyclical graph of network steps. This forms

the topology of the network architecture, and is how hardware functions to process data, since layer inputs depend on the output of other layers. Thus, by casting individual layers as graph nodes, connected to other nodes by the flow of logits in the network, we can isolate the span of signals necessary to recover a network. Namely, looking at nodes or edges suffices to recover individual steps of the network, which combine in sequential and semantic order as a neural network.

Classifying steps of a model entails converting a time-series signal to a series of labeled operations. The EM signal responds only to the GPU's instantaneous performance, but because the GPU executes a neural network sequence, there is rich context in both the window before and after any one segment of the signal. Some steps are often followed by others, such as pooling operations after a series of convolutions. We take advantage of this bidirectional context of our signal to sequence classification problem by utilizing a recurrent neural network to classify the observed signal.

Bidirectional Long Short-Term Memory (BiLSTM) networks are well-suited for processing time-series signals [16]. We train a two-layer BiLSTM network to classify each signal sample into a predicted step (see 4.2-b). The input to our network is a sliding window of the time-series signal, the entirety of which is classified according to the step operations from our simulated network architecture dataset. We train the BiLSTM by minimizing the standard cross-entropy loss between the predicted per-sample labels and the ground-truth labels.

This approach proves robust at identifying the sequence of steps. It enables all of our experiments and all GPU's tested to recover the layers of the target network, including their *type* (e.g., fully connected, convolution, recurrent, etc.), activation function, and any subsequent forms of pooling or batch normalization. What remains is to recover layer hyperparameters.

Hyperparameter Estimation

The number of hyperparameters that describe a layer type depends on its linear step. For instance, a CNN layer type's linear step is described by size, padding, kernel size, number of channels, and stride hyperparameters. Hyperparameters within a layer must be *intra-consistent*. Of the six CNN hyperparameters (stride, padding, dilation, input, output, and kernel size), any one is determined by the other five. Hyperparameters must also be *inter-consistent* across consecutive layers: the output of one layer must fit

the input of the next. A brute-force search of consistent hyperparameters easily becomes intractable for deeper networks; we therefore first estimate hyperparameters for each layer in isolation, and then jointly optimize to obtain consistency.

Initial estimation. We estimate a specific hyperparameter of a specific layer type, by pretraining a DNN. We pretrain a suite of such DNNs, one for each (layer type, hyperparameter) pairing. With the layers (and their types) recovered, we can estimate each hyperparameter using these pretrained (layer type, hyperparameter) recovery DNNs.

These DNN accept feature vectors describing two signal segments: the linear step and the immediately subsequent step. The subsequent step (e.g., activation, pooling, batch normalization) requires effort proportional to the linear step's output dimensions, thus its inclusion informs the estimated output dimension. Each segment's features involves extracting averages from **(i)** partitioning the segment uniformly into N windows, and **(ii)** concatenating the segment's time duration. However, there is no guarantee that this produces a valid network in that neighboring step estimates are consistent.

Joint optimization. To enforce consistency on initial estimates we jointly optimize, seeking values that *best fit their initial estimates, subject to consistency constraints*. Our optimization minimizes the convex quadratic form

$$\min_{x_i \in \mathbb{Z}^{0+}} \sum_{i \in \mathcal{X}} (x_i - x_i^*)^2 \ , \quad \text{subject to consistency constraints,} \tag{4.1}$$

where $\mathcal{X}$ is the set of all hyperparameters across all layers; x_i^* and x_i are the initial estimate and optimal value of the i-th hyperparameter, respectively. The imposed consistency constraints are:

(i) The output size of a layer agrees with the input size of the next layer.

(ii) The input size of the first layer agrees with the input feature size.

(iii) The output size of a CNN layer does not exceed its input size.

(iv) The hyperparameters of a CNN layer satisfy

$$s_{\text{out}} = \left\lfloor \frac{s_{\text{in}} + 2\beta - \gamma(k-1) - 1}{\alpha} + 1 \right\rfloor , \tag{4.2}$$

where α, β, γ, and k denote the layer's stride, padding, dilation, and kernel size, respectively.

(v) Heuristic constraint: the kernel size must be odd.

Among these constraints, **(i-iii)** are linear constraints, which preserves the convexity of the problem. The heuristic **(v)** can be expressed as a linear constraint: for every kernel size parameter k_j, we introduce a dummy variable τ_j, and require $k_j = 2\tau_j + 1$ and $\tau_j \in \mathbb{Z}^{0+}$. Constraint **(iv)** , however, is troublesome, because the appearance of stride α and dilation γ, both of which are optimization variables, make the constraint nonlinear.

Since all hyperparameters are non-negative integers, the objective must be optimized via integer programming: IP in general case is NP-complete [37], and the nonlinear constraint **(iv)** does not make life easier. Fortunately, both α and γ have very narrow ranges in practice: α is often set to be 1 or 2, and γ is usually 1, and they rarely change across all CNN layers in a network. As a result, they can be accurately predicted by our DNN models; we therefore retain the initial estimates and do not optimize for α and γ, rendering (4.2) linear. Even if DNN models could not reliably recover α and γ, one could exhaustively enumerate the few possible α and γ combinations, and solve the IP problem (4.1) for each combination, and select the best recovery.

The IP problem with a quadratic objective function and linear constraints can be easily solved, even when the number of hyperparameters is large (e.g., $> 1,000$). In practice, we use IBM CPLEX [6], a widely used IP solver. Optimized hyperparameters remain close to the initial DNN estimates, yet differ in that they are guaranteed to define a valid network structure.

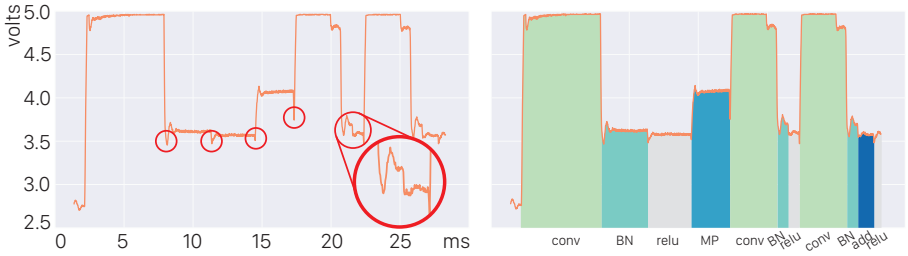

Figure 4.2: **Leaked magnetic signal.** (left) Our induction sensor captures a magnetic signal of a CNN running on the GPU. The GPU has to synchronize steps, resulting in a sharp drop of the signal level (highlighted by selected red circles). (right) We can accurately classify the network steps and reconstruct the topology, as indicated by the labels under the x-axis. Here we highlight the signal regions associated with convolutions (conv), batch-norm (BN), Relu activations (relu), max-pooling (MP), and adding steps together (add).

4.2 Evaluation

Data capture. Pretraining the recovery DNN models (recall 4.1) requires an annotated dataset with pairwise correspondence between signal and step types (see 4.1). We can automatically generate an annotated signal for a given network and specific GPU hardware, simply by executing a query (with arbitrary input values) on the GPU to acquire the signal. Timestamped ground-truth GPU operations are available by deep learning libraries (e.g., `torch.autograd.profiler` in PyTorch and `tf.profiler` in TensorFlow).

Training Set Details. The set of networks to be annotated could in principle consist **(i)** solely of randomly generated networks, on the basis that data values and "functionality" are irrelevant to us, and the training serves to recover the substeps of a layer; or **(ii)** of curated networks or those found in the wild, on the basis that such networks are more indicative of the typical black-box. We construct our training set as a mixture of both approaches.

Randomly generated networks involve base steps made up of a mixture of fully-connected, recurrent, and CNN layers. These are accompanied by 5 different activation functions, 2 types of pooling layers, and a potential normalization operation. Off the shelf networks consist of VGG and ResNet variants. All in all we consider 500 networks for training, ranging from 4 to 512 steps per network and culminating in $70,933$ individual steps in total. When we construct these networks, their input image resolutions are randomly chosen from [224×224, 96×96, 64×64, 48×48, 32×32]: the highest resolution is used in ImageNet, and lower resolutions are used in datasets such as CIFAR.

4.3 Experiments & Results

Topology reconstruction. As discussed in 4.1, we use a BiLSTM model to predict the network step for each single sample. Table 4.1 reports its accuracy, measured on an Nvidia Titan V GPU. There, we also break the accuracy down into measures of individual types of network steps, with an overall accuracy of **96.8%**. An interesting observation is that the training and test datasets are both unbalanced in terms of signal samples (see last column of Table 4.1). This is because in practice convolutional layers are computationally the most expensive, while activation functions and pooling

Table 4.1: Classification accuracy of network steps (Titan V)

Layer Type	Prec.	Rec.	F1	# samples
LSTM	.997	.992	.995	8,704
Conv	.993	.996	.994	447,968
Fully-connected	.901	.796	.846	10,783
Add	.984	.994	.989	22,714
BatchNorm	.953	.955	.954	47,440
MaxPool	.957	.697	.806	4,045
AvgPool	.371	.760	.499	675
ReLU	.861	.967	.911	28,512
ELU	.464	.825	.594	2,834
LeakyReLU	.732	.578	.646	9,410
Sigmoid	.694	.511	.588	8,744
Tanh	.773	.557	.648	4,832
Weighted Avg.	**.968**	**.967**	**.966**	-

are lightweight. Also, steps like average pooling are less frequently used. While such data imbalance does reflect reality, when we use them to train and test, most of the misclassifications occur at those rarely used, lightweight network steps, whereas the majority of network steps are classified correctly.

We evaluate the quality of topology reconstruction using normalized Levenshtein edit distance that has been used to evaluate network structure similarity [17, 20]. Here, Levenshtein distance measures the minimum number of operations—including adding/removing network steps and altering step type—needed to fully rectify a recovered topology. This distance is then normalized by the total number of steps of the target network.

Among the 64 tested networks, 40 of the reconstructed networks match precisely their targets, resulting in *zero* Levenshtein distance. The average normalized Levenshtein distance of all tested networks is **0.118**, and confirms our networks are recovered with often exact step matches and model lengths.

To provide a sense of how the normalized Levenshtein distance is related to a network's ultimate performance, we conduct an additional experiment to gauge reconstruction quality via classification accuracy. We consider AlexNet (referred as model A) and its five variants (refered as model B, C, D, and E, respectively). The variants are constructed by randomly altering some of the network steps in model A. The Levenshtein distances between model A and its variants are 1, 2, 2, 5, respectively, and the normalized Levenshtein distances are 0.05, 0.11, 0.11, 0.28 (see Fig. A.2). We then measure the performance (i.e., standard test accuracy) of these models on CIFAR-10. As the edit distance increases, the model's performance drops.

Table 4.2: Model extraction accuracy on CIFAR-10

Model	Target	Titan V	Titan X	GTX1080	GTX960
VGG-11	89.03	89.61	89.63	88.46	88.3
VGG-16	90.95	91.08	91.03	89.33	90.78
AlexNet	81.68	85.26	85.11	85.27	85.03
ResNet-18	92.77	92.61	92.82	92.79	92.04
ResNet-34	92.21	92.28	92.95	90.81	92.71
ResNet-50	90.89	91.8	91.97	91.2	91.29
ResNet-101	91.58	91.91	91.85	91.37	91.72

DNN hyperparameter estimation. Next, we report the test accuracies of our DNN models (discussed in Ch. 4.1) for estimating hyperparameters of convolutional layers. Our test data here consists of 1804 convolutional layers. On average, our DNN models have **96%-97%** accuracy. The breakdown accuracies for individual hyperparameters are shown in Table A.1 of the appendix.

Reconstruction quality measured as classification accuracy. Ultimately, the reconstruction quality must be evaluated by how well the reconstructed network performs in the task that the original network aims for. To this end, we test seven networks, including VGGs, AlexNet, and ResNets, that have been used for CIFAR-10 classification (shown in Table 4.2). We treat those networks as black-box models and reconstruct them from their magnetic signals. We then train those reconstructed networks and compare their test accuracies with the original networks' performance. Both the reconstructed and original networks are trained with the same training dataset for the same number of epochs. The results in Table 4.2 show that for all seven networks, including large networks (e.g., ResNet101), the reconstructed networks perform almost as well as their original versions.

Transfer Attacks. Results of using such an approach to enable transfer attacks is deffered to Appendix A. Further findings discussing extraction and comparisons among GPUs can be found in the full text [34].

Chapter 5

Data-Driven Hair Contact

Modern hair simulation pipelines are largely throttled by their handling of inter-strand contact. Accurate collision resolution between rods is computationally expensive at scale and virtually prohibitive at large time steps. The overall difficulty stems from time required to converge on a solution to collision handling when treating large intertwined contact networks. It is often necessary to approximate the contact handling in order to maintain simulation progress. Concessions may involve adopting a simplified contact model, curtailing the number of contacts treated, or limiting the time allotted towards contact resolution (often by capping the number of solver iterations permitted for friction handling). The are no guarantees on the accuracy or convergence of the resulting contact update that shapes each timestep of most practically sized examples. Yet, even after surrendering accuracy, contact remains a bottleneck. A technique for efficient large scale hair contact resolution that addresses these shortcomings is therefore highly desirable.

We propose to continue the partnership developed between simulated examples and graph formulations in order to tackle the challenges surrounding efficiently simulating hair contact. We will generate strand-strand contact resolution data to pair with input collision configurations by running a baseline hair simulation on desired examples. Inspecting these samples we discover time-varying contact clusters that form spatially, and their properties that inform the contact solve. We translate these contact clusters into graphs that can be used to train a machine learning model to infer post-solve degrees of freedom from contact configurations. We propose to utilize these graphs in conjunction with aggregated simulation data to train an efficient approximate hair contact model that can be used to exhibit large scale simulations.

Objective We are motivated to introduce a data-driven alternative that can expedite contact for hair simulations. However, data-driven approximations face two concerns when applied to hair contact. Firstly, elastic rods do not map naturally to existing network architectures or model reduction techniques. Secondly, the space of contact configurations we aim to approximate grows combinatorially. We seek a result free of visual artifacts and flexible enough to work with various strand models. Any method is acceptable so long as it can achieve similar results and reduce the costs incurred by the simulation.

5.1 Method

We propose to train a data-driven contact resolution model based on slow albeit accurate collision solves, so that the data used to guide our model is physically principled. This allows us to leverage existing discrete elastic rod simulations that we wish to emulate with the ability to generate labeled datasets as needed. The simulated input configurations and output collision resolutions generated are used to inform a neural network, making us the first to bridge physics-based elastic rod simulations with machine learning.

We propose to address both the structuring and constraining of our data pairs through the use of Graph Neural Networks [2, 4, 30, 31, 38, 41, 42, 43]. We reformulate the data generated by mapping contact clusters of strands onto a graph structure. We propose a novel interpretation of dynamic piecewise linear elastic curves in 3D as embeddings held in nodes and edges of a graph network. Elements in the graph network are combined with machine learning algorithms to process latent embeddings and to communicate across the graph which includes contact edges. This transformation sidesteps the overwhelming complexity of data introduced by simulating free curves in space with possibly varying degrees of freedom and contact stencils. This framework allows us to moderate any exponential or combinatorial expansion of our dataset, and constrains our efforts to solve local operators on the proposed graph. The resulting simulation targets foremost the improvement in efficiency of collision resolution while remaining visually plausible.

5.2 Contributions

Our proposed approach features the following contributions:

- Fast large scale simulations trained from small scenes
- Stable results free of artifacts not present in training
- First to combine hair contact with machine learning
  - A novel Graph Network processing formulation and application
- Flexibility to replace any black box strand contact resolution scheme
- Model and parameter agnostic, working for straight and curly strands

5.3 Setup

The nonlinear number of collisions and changing contact bodies poses a challenge to training a neural model which requires consistently shaped inputs. Penetrating objects may be composed of different degrees of freedom and shapes, making a fully connected multi-layer perceptron or convolutional network difficult to assemble. The elements in contact also change as collisions appear and are resolved throughout the simulation, with no spatial-temporal coherence. In order to manage variable sized inputs pertaining to contact clusters and strands that change in the number of primitives involved, we take advantage of Graph Neural Networks.

Hair strands are formed of vertices connected by edges, and these edges come into contact with other strands at their corresponding edges (i.e. edge-edge collision detection). Therefore, rather than directly translating hair vertices and strand edges to graph nodes and edges, we stand to benefit from viewing the *dual* of our rod structure.

We map the material edges of each strand to graph nodes. Consequently, we connect these nodes with graph edges that represent the internal material vertices and edges of each strand. This enables us to treat interactions between edges, i.e. contacts, in simulation space by connecting nodes with an edge in our graph formulation. We categorize these volatile connections as contact edges, and together with the nodes and internal-vertex edges they form the DualGraph.

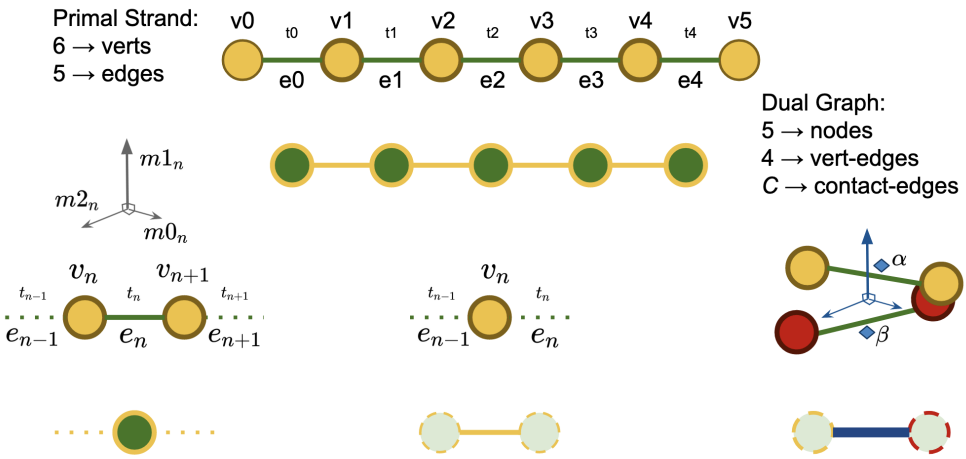

Figure 5.1: **Dual Strand.** We map elastic rods to a neural-amenable graph of their dual representation by converting simulation edges to GraphNet nodes and connecting them with graph edges.

This inverted approach has several advantages. Firstly, contacts between strands can be simply connected via a conventional edge between nodes in the DualGraph, avoiding the redundancies and intricacies associated with relating interacting edges in the primal picture. Furthermore, discrete elastic rods exhibit twist and bending modes that are only present at internal vertices of the strand. Twisting and bending can be thought of as deviations to material frames present on strand edges, and so a DualGraph framework more naturally captures these features.

Once constructed, we convert the features hosted by our DualGraph by using the encoder-processor-decoder framework adopted by previous GraphNet simulation frameworks [38, 42, 41]. The first step converts features into an in-place latent embedding on the graph. Next, we process our graph to relate information across adjacent and incident graph elements (e.g. inform nodes with incident edges). Lastly, we decode the graph back to physical quantities for use in our simulation. In a force or contact-only based formulation, we may decode contact edges to retrieve impulses. If velocities are the desired output than we convert graph nodes and material edges into the vectors of similar degrees of freedom that indicate changes to velocities for each strand or vertex.

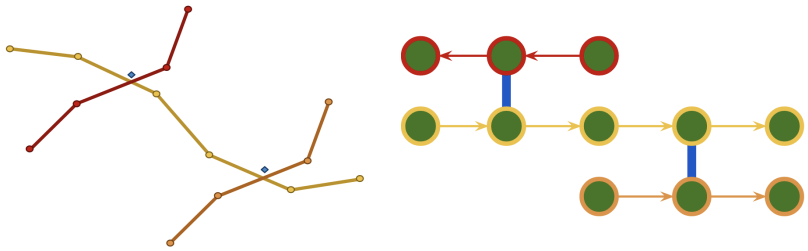

Figure 5.2: **Three strands in contact.** Strands in contact (left) naturally form DualGraphs (right), where edge-edge collisions are easily represented.

Data Aggregation We take our data from the readily available and popularly used friction and impulse formulation of hair contact provided by discrete elastic rods. This approach has traditionally been shown at scale [24, 7] and suffers from the common contact handling bottlenecks we aim to address. Other penalty-based variants [13, 29] may also be used.

Training from an entirely simulated examples allows us to benefit through principles of *Direct Policy Learning*. Rather than relying on a predetermined or constrained dataset, we instead have an interactive demonstrator by means of our physically based target environment. This expert oracle can be used to provide feedback on predicted rollout trajectories and demonstrations; and means we can gather new data and evolve the training dataset.

This form of *Imitation learning* proves particularly fruitful given the configuration of our simulation state across timesteps are not independent and identically distributed from one another. The output of one timestep (influenced by an inference) will color the input to the next timestep. Therefore any approximation error may lead us towards an unfamiliar input space, and this error will accumulate as we strive to predict from only the familiar that the network used for training.

Consequently, rather than anticipating all the data necessary for training, it is easier for an 'expert' to demonstrate the target. By utilizing the oracle to tame a possibly diverging input space, errors neither accumulate and the agent avoids places the expert never visited.

To apply this approach, we start with (1) a DualGraph trained on initial principled hair contact demonstrations. Then, we execute the following loop until we converge. In each iteration, we (2) collect trajectories by solving contacts via a DualGraph (which we obtained in the previous iteration) and using to simulate the strands at each timestep. Then, for every inference, we

(3) collect ground-truth output from the oracle (what would have he done in the same configuration). Finally, we (1) train a new DualGraph policy using this feedback.

5.4 Evaluation

The desired output of a hair contact pipeline is the efficient and accurate modeling of strand-strand behavior. We will show that by using DualGraphs we are able to reduce the cost of collision resolution in hair simulations without introducing visual artifacts. Our approach will be the first to combine strand collision resolution with data-driven methods, and stands to benefit from the limitless data generation that our base simulation provides. If successful, we will not only introduce an avenue for faster large scale hair simulations, but also pave the way for future data driven explorations relating to hair, thin structures, and other physics-based phenomena.

Figure 5.3: **Hairball.** We aim to approximate grooms generated by state-of-the-art hair simulations [24] at a fraction of the computational cost.

We are motivated by early successes in replacing contact solves with inferred approximations. Through a mixture of DualGraph and careful data aggregation, we have been able to simulate piles of strands coming into stable resting contact. We next seek to evaluate the robustness of our method to larger scale examples. We aim to explore larger hairball examples (see Fig. 5.3) and plan to show how a small but representative contact scenario can inform a groom several orders of magnitude higher. Once a stable hairball featuring thousands of strands can be demonstrated, we will aggregate timing comparisons in order to submit our findings.

Chapter 6

Timeline

Our proposed methodology calls for utilizing datasets of synthetic data to learn realistic behavioral models via graph algorithms. Simulated examples more easily provide access to labeled and controlled behaviors and interactions. Graph formulations then constrain the space of digital features by simplifying efforts to operations on nodes and edges of the formulated graph.

We propose to validate our methodology by studying its implications on additive manufacturing and computer vision objectives (Ch. 3), physical side channel attacks on modern GPUs of black-box neural architectures (Ch. 4), and resolving bottlenecks in state-of-the-art strand-strand contact simulations (Ch. 5). We aim to show that, compared to alternative methods, graph-based handling of data driven models outperforms existing works in terms of robustness and quality guarantees.

The timeline for completing the proposed dissertation is as follows:

- ☑ LayerCodes - Fall 2019
- ☑ Neural Snooping - Fall 2021
- ☐ Thesis Proposal - Winter 2021/2022
- ☐ Data-Driven Hair Contact Submission - Spring 2022
- ☐ Thesis Writing - Spring 2022
- ☐ Thesis Defense - Summer 2022

Thus I plan to defend my thesis by the end of this calendar school year.

Chapter 7

References

[1] Lejla Batina et al. "CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel". In: *Proceedings of the 28th USENIX Security Symposium*. USENIX Association. 2019.

[2] Peter W. Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *CoRR* abs/1806.01261 (2018). arXiv: 1806.01261. URL: http://arxiv.org/abs/1806.01261.

[3] Berk Calli et al. "The YCB object and Model set: Towards common benchmarks for manipulation research". In: *Advanced Robotics (ICAR), 2015 International Conference on*. IEEE. 2015, pp. 510–517.

[4] Michael B Chang et al. "A Compositional Object-Based Approach to Learning Physical Dynamics". In: *arXiv preprint arXiv:1612.00341* (2016).

[5] Boyuan Chen et al. "The Boombox: Visual Reconstruction from Acoustic Vibrations". In: *CoRR* abs/2105.08052 (2021). arXiv: 2105.08052. URL: https://arxiv.org/abs/2105.08052.

[6] IBM ILOG Cplex. "V12. 1: User's Manual for CPLEX". In: *International Business Machines Corporation* 46.53 (2009), p. 157.

[7] Gilles Daviet. "Simple and Scalable Frictional Contacts for Thin Nodal Objects". In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392439. URL: https://doi.org/10.1145/3386569.3392439.

[8] Ambra Demontis et al. "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks". In: *28th USENIX Security Symposium Security 19)*. 2019, pp. 321–338.

[9] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[10] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[11] Dave Epstein, Boyuan Chen, and Carl Vondrick. "Oops! Predicting Unintentional Action in Video". In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[12] Yun (Raymond) Fei et al. "A Multi-scale Model for Simulating Liquid-fabric Interactions". In: *ACM Trans. Graph.* 37.4 (Aug. 2018), 51:1–51:16. ISSN: 0730-0301. DOI: 10.1145/3197517.3201392. URL: http://doi.acm.org/10.1145/3197517.3201392.

[13] Yun (Raymond) Fei et al. "A Multi-scale Model for Simulating Liquid-hair Interactions". In: *ACM Trans. Graph.* 36.4 (July 2017), 56:1–56:17. ISSN: 0730-0301. DOI: 10.1145/3072959.3073630. URL: http://doi.acm.org/10.1145/3072959.3073630.

[14] Corey Goldfeder and Peter K Allen. "Data-driven grasping". In: *Autonomous Robots* 31.1 (2011), pp. 1–20.

[15] Corey Goldfeder et al. "The columbia grasp database". In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE. 2009, pp. 1710–1716.

[16] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. "Bidirectional LSTM networks for improved phoneme classification and recognition". In: *International Conference on Artificial Neural Networks*. Springer. 2005, pp. 799–804.

[17] Alex Graves et al. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 369–376.

[18] J. Harris, J.L. Hirst, and M. Mossinghoff. *Combinatorics and Graph Theory*. Undergraduate Texts in Mathematics. Springer New York, 2009. ISBN: 9780387797113. URL: https://books.google.com/books?id=DfcQaZKUVLwC.

[19] K. He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[20] Xing Hu et al. "DeepSniffer: A DNN Model Extraction Framework Based on Learning Architectural Hints". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2020, pp. 385–399.

[21] Wenzel Jakob. *Mitsuba renderer*. http://mitsuba-renderer.org. 2010.

[22] K. Jo, M. Gupta, and S.K. Nayar. "DisCo: Display Camera Communication Using Rolling Shutter Sensors". In: *ACM Trans. on Graphics (also Proc. of ACM SIGGRAPH)* 35.5 (July 2016), 150:1–13.

[23] Daniel Kappler, Jeannette Bohg, and Stefan Schaal. "Leveraging big data for grasp planning". In: *ICRA*. IEEE. 2015, pp. 4304–4311.

[24] Danny M. Kaufman et al. "Adaptive Nonlinearity for Collisions in Complex Rod Assemblies". In: *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601100. URL: https://doi.org/10.1145/2601097.2601100.

[25] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721. eprint: https://doi.org/10.1177/0278364913495721. URL: https://doi.org/10.1177/0278364913495721.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[27] D. Li et al. "AirCode: Unobtrusive Physical Tags for Digital Fabrication". In: *ACM Symposium on User Interface Software and Technology (UIST)*. Oct. 2017.

[28] Dingzeyu Li et al. "Acoustic Voxels: Computational Optimization of Modular Acoustic Filters". In: *ACM Trans. Graph.* 35.4 (2016).

[29] Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. *Codimensional Incremental Potential Contact*. 2020. arXiv: 2012.04457 [cs.GR].

[30] Yunzhu Li et al. "Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=rJgbSn09Ym.

[31] Yunzhu Li et al. "Propagation Networks for Model-Based Control Under Partial Observation". In: *ICRA*. 2019.

[32] Marco Livesu et al. "From 3D models to 3D prints: an overview of the processing pipeline". In: *Comput. Graph. Forum* 36.2 (2017), pp. 537–564.

[33] Aleksander Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rJzIBfZAb.

[34] Henrique Teles Maia et al. *Can one hear the shape of a neural network?: Snooping the GPU via Magnetic Side Channel*. 2021. arXiv: 2109.07395 [cs.CR].

[35] Henrique Teles Maia et al. "LayerCode: Optical Barcodes for 3D Printed Shapes". In: *ACM Trans. Graph.* 38.4 (July 2019), 112:1–112:14. ISSN: 0730-0301. DOI: 10.1145/3306346.3322960. URL: http://doi.acm.org/10.1145/3306346.3322960.

[36] Nasser M Nasrabadi. "Pattern recognition and machine learning". In: *Journal of electronic imaging* 16.4 (2007), p. 049901.

[37] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[38] Tobias Pfaff et al. "Learning Mesh-Based Simulation with Graph Networks". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=roNqYL0_XP.

[39] Ben Redwood, Filemon Schffer, and Brian Garret. "The 3D Printing Handbook: Technologies, design and applications". In: (2017).

[40] Joseph M Romano et al. "Human-inspired robotic grasp control with tactile sensing". In: *IEEE Transactions on Robotics* 27.6 (2011), pp. 1067–1079.

[41] Alvaro Sanchez-Gonzalez et al. "Graph Networks as Learnable Physics Engines for Inference and Control". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, July 2018, pp. 4470–4479. URL: http://proceedings.mlr.press/v80/sanchez-gonzalez18a.html.

[42] Alvaro Sanchez-Gonzalez et al. "Learning to Simulate Complex Physics with Graph Networks". In: *International Conference on Machine Learning*. 2020.

[43] Franco Scarselli et al. "The Graph Neural Network Model". In: *Trans. Neur. Netw.* 20.1 (Jan. 2009), pp. 61–80. ISSN: 1045-9227. DOI: 10.1109/TNN.2008.2005605. URL: https://doi.org/10.1109/TNN.2008.2005605.

[44] Adriana Schulz et al. "Interactive Design Space Exploration and Optimization for CAD Models". In: *ACM Transactions on Graphics* 36.4 (July 2017).

[45] Adriana Schulz et al. "Interactive Exploration of Design Trade-Offs". In: *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: 10.1145/3197517.3201385. URL: https://doi.org/10.1145/3197517.3201385.

[46] Adriana Schulz et al. "Interactive robogami: An end-to-end system for design of robots with ground locomotion". In: *The International Journal of Robotics Research* 36.10 (2017), pp. 1131–1147.

[47] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[48] Daniel Sims et al. *Stretchcam: Zooming Using Thin, Elastic Optics*. Tech. rep. Dec. 2017.

[49] Peter Teufl, Udo Payer, and Guenter Lackner. "From NLP (Natural Language Processing) to MLP (Machine Language Processing)". In: *Computer Network Security*. Ed. by Igor Kotenko and Victor Skormin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 256–269. ISBN: 978-3-642-14706-7.

[50] Jacob Varley et al. "Shape Completion Enabled Robotic Grasping". In: *Intelligent Robots and Systems (IROS), IEEE/RSJ 2017 International Conference on*. extended version preprint at arXiv:1609.08546.

[51] Hongyi Xu et al. "Interactive Material Design Using Model Reduction". In: *ACM Trans. Graph.* 34.2 (Mar. 2015). ISSN: 0730-0301. DOI: 10.1145/2699648. URL: https://doi.org/10.1145/2699648.

[52] Xiaojun Xu et al. "Neural Network-Based Graph Embedding for Cross-Platform Binary Code Similarity Detection". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 363–376. ISBN: 9781450349468. DOI: 10.1145/3133956.3134018. URL: https://doi.org/10.1145/3133956.3134018.

[53] Qingnan Zhou and Alec Jacobson. "Thingi10K: A Dataset of 10,000 3D-Printing Models". In: *arXiv preprint arXiv:1605.04797* (2016).

Appendix A
Supplemental Findings

Figure A.1: **Complex shapes.** A peek into the diversity of tested shapes within our database. Each view presented is correctly decoded by our graph-based algorithm, including those with bumpy, shell, thin, curvy, and other challenging properties.

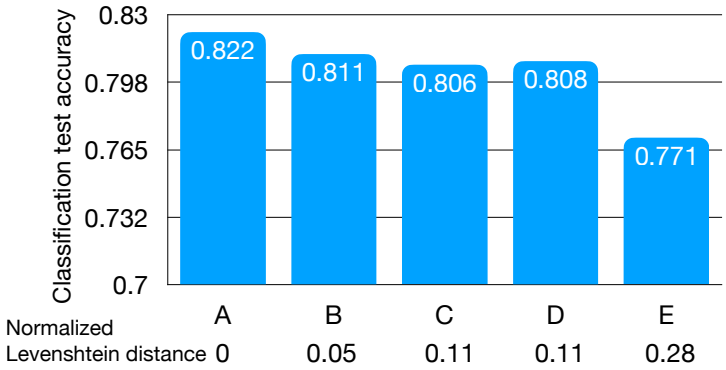

Figure A.2: Each model's classification accuracy drops as its Levenshtein distance from the original model (model A: AlexNet) increases.

Table A.1: **DNN estimation accuracies.** Using the 1804 convolutional layers in our test dataset, we measure the accuracies of our DNN models for estimating the convolutional layers' hyperparameters. Here, we break the accuracies down into the accuracies for individual hyperparameters.

	Kernel	Stride	Padding	Image-in	Image-out
Precision	0.971	0.976	0.965	0.968	0.965
Recall	0.969	0.975	0.964	0.969	0.968
F1 Score	0.969	0.975	0.962	0.967	0.965

Transfer Attack

An adversarial transfer attack attempts to design an input that tricks an unknown target model. The name *transfer* alludes to the method of attack: The attacker builds a surrogate, an approximation (informed guess) of the unknown target model, and seeks out an input that tricks the surrogate. The attacker hopes that the exploit "transfers" to the actual target, i.e., that an input that tricks the surrogate also tricks the target. The likelihood of a successful attack increases as the surrogate better approximates the target. In a black-box setting, finding an effective surrogate is very hard [8]. Therefore, the attacker wishes to design a more informed surrogate. One avenue toward this is to design surrogates with topology and parameters similar to the target.

CIFAR-10 Dataset. Here we test on six networks found in the wild, ranging from VGGs to AlexNet to ResNets, as listed on the header row of A.2. The table shows the percent of successful transfer attacks over $5,000$ attempts on the CIFAR-10 dataset.

We consider each target architecture on each of four GPUs in turn (top four rows of A.2). We consider each such architecture-GPU combination, in turn, as black-box target. Using the side channel exploit, we reconstruct the target's structure to obtain a surrogate architecture, which we train on CIFAR-10 to obtain a surrogate model. We craft inputs that trick the surrogate, and evaluate whether those inputs also trick the target. Transfer attack success is defined as the percent of generated inputs (based on the surrogate) that correctly cause the trained target model to mislabel an input. All adversarial inputs are generated via Projected Gradient Descent [33], using an ϵ of 0.031 and an α of 0.003 for all results. The success rate of the transfer attacks is summarized in the upper four rows of A.2.

To gauge the success rates of the "side channel surrogates," we compare them against "white-box surrogates." We build six white-box surrogates, corresponding to the six known target architectures; these white-box surrogates differ only in weights, as the surrogates are trained from scratch on CIFAR-10. The idealized white-box surrogates serve as a benchmark for effective surrogates; refer to the success rates in the bottom six rows of A.2.

Remarkably, the "side-channel surrogates" offer comparable success rates to "white-box surrogates." The relative success of side-channel surrogates becomes more pronounced for deeper networks (ResNets), where it appears that architecture dominates sensitivity to weight values. When the number of layers is small, as in VGG-11 and AlexNet architectures, the margin for error decreases, and more importance is given to the weights of the target. However, even in these cases where attack performance drops, the side-channel surrogates closely match the success rate of their white-box counterparts, displayed in the lower rows. In other words, the side-channel reconstruction effectively turns a black-box into a white-box attack.

MNIST Dataset. Similar to our analysis of CIFAR-10 transfer attacks, we also conduct transfer attack experiments on the MNIST dataset. We download four networks online, which are not commonly used. Two of them are convolutional networks (referred as CNN1 and CNN2), and the other two are fully connected networks (referred as DNN1 and DNN2). None of these

Table A.2: Transfer attack results on CIFAR-10.

		Target Model					
		ResNet-18	ResNet-34	ResNet-101	VGG-11	VGG-16	AlexNet
Source Model	GTX-960	98.56	92.51	91.20	63.41	72.57	58.90
	GTX-1080	97.88	90.86	86.24	64.69	55.19	56.83
	Titan X	98.32	93.45	84.47	61.89	77.36	68.41
	Titan V	98.48	93.65	91.27	64.39	72.77	60.17
	ResNet-18	97.70	90.72	80.27	47.98	86.64	30.56
	ResNet-34	97.21	92.46	82.30	51.42	85.60	32.34
	ResNet-101	92.53	86.98	92.95	53.98	83.04	30.55
	VGG-11	65.86	57.82	57.52	60.24	65.50	39.95
	VGG-16	74.00	61.54	54.23	41.60	74.29	29.57
	AlexNet	10.11	9.59	10.19	11.60	10.42	62.70

Table A.3: Transfer attack results on MNIST.

		Target Model			
		CNN1	CNN2	DNN1	DNN2
Source Model	GTX-1080	.802	.878	.999	.874
	CNN1	.858	.226	.785	.476
	CNN2	.395	.884	.354	.354
	DNN1	.768	.239	.999	.803
	DNN2	.703	.219	.975	.860

networks appear in the training dataset. We treat these networks as black-box targets, reconstruct a side-channel surrogate for each, and attack the four targets; results are shown in A.3. As baselines, we also train white-box surrogates with the exact architecture of the four target models.

All four of our extracted networks, visible in the top row of A.3 achieve high transfer attack scores against our candidate targets. These high scores suggest a close approximation of the target models by our reconstructed networks. The similarity between our extracted network's transfer attack results and the results achieved by the matching source model across the bottom four rows also indicates a strong correspondence in the achieved architectures. We find that even across the MNIST dataset we are able to generate a model that behaves akin to a white-box transfer attack.