Predicting Static Stability with Data-Driven Physics in Air Cargo Palletizing

Philipp Gabriel Mazur
University of Cologne, Germany, mazur@wim.uni-koeln.de

No-San Lee
University of Cologne, Germany, lee@wim.uni-koeln.de

Janik Euskirchen
University of Cologne, Germany, jeuskirchen@protonmail.com

Detlef Schoder
University of Cologne, Germany, schoder@wim.uni-koeln.de

Predicting Static Stability with Data-Driven Physics in Air Cargo Palletizing

Philipp Gabriel Mazur[1], No-San Lee[1], Janik Euskirchen[2], and Detlef Schoder[1]

University of Cologne, Cologne Institute for Information Systems, Cologne, Germany
[1] {mazur,lee,schoder}@wim.uni-koeln.de
[2] {jeuskir1}@smail.uni-koeln.de

Abstract. Proposing air cargo palletizing solutions requires an assessment by a physics engine of whether a solution is physically stable, which can take up a disproportionate amount of computation and, thus, produce a bottleneck in the optimization pipeline. This problem can be tackled by replacing the physics engine with a data-driven model that learns to map proposed packing pattern solutions directly to its stability value. We develop a prototype of such a data-driven model and find that this approach yields practicable results and does so multiple orders of magnitudes faster than a commonly used physics engine.

Keywords: machine learning, data science, data-driven physics, air cargo palletizing, static stability simulation

1 Introduction

In air cargo palletizing, items must be arranged within a given volume of space, either on top of a pallet or within a container. To maximize load factor and priority of cargo operations, researchers make use of optimization methods such as genetic algorithms (GA) to find optimal arrangements (*packing patterns*). From a research perspective, the challenge to find practicable if not optimal packing patterns is known as the pallet loading problem (PLP) or container loading problem (CLP). It defines a np-hard and highly complex problem space. To tackle the problem space, information systems (IS) deploy exact or approximative algorithms, e.g. placement heuristics, and improvement heuristics to find good solutions [1, 2]. GAs are one of the most important heuristics and frequently act as optimization heuristics [3–5].

Practical packing patterns need to satisfy a variety of further constraints. This work considers in particular the physical constraints of stability [1, 6, 7]. Previous works distinguish between static and dynamic stability. A packing pattern is *statically* stable if the items are not displaced while being loaded onto unit load devices (ULD). A packing pattern is *dynamically* stable if items are not displaced while the ULD is intentionally being moved during transport, e.g. during a flight [1]. ULDs are standardized pallets and containers that are used for transporting cargo in aircrafts and vary in size, volume, weight, and shape. One way to find out if a proposed packing pattern solution is viable with respect to stability, is to simulate the physical behavior of the proposed pattern with a physics engine [7, 8].

Physical simulations are costly to employ and require substantial amounts of computing power. Especially when embedded in time-sensitive applications such as optimization pipelines (e.g., heuristic search processes using GA and considering numerous constraints [5, 9]), or combined problems such as vehicle routing and container loading problems (3L-CVRP) [10], this negatively impacts overall systems performance [5, 6, 11]. Due to the number of simulations during optimization, simulations act as a bottleneck. It is therefore desirable to circumvent this bottleneck to speed up the process of finding packing patterns that satisfy all constraints without losing the flexibility and adaptability of physical simulations to varying problem cases.

Therefore, to tackle the previously mentioned challenge, we employ a machine learning (ML) approach. ML is a subfield of artificial intelligence that deals with creating models whose behavior is extracted from data rather than being explicitly defined by the programmer [12]. Neural networks (NN) are a ML model class that map features to target predictions by applying a series of non-linear functions. The NN parameters are fitted to a given training dataset by minimizing the average loss between prediction and target values, such that the model optimally maps input features to targets [12]. Holden et al. [13] and Wiewel et al. [14] among others, have demonstrated the feasibility of employing a data-driven model to gain a significant speed-up over regular physics engines. The authors first project a physical state into a lower-dimensional "latent space" and then predict the next latent vector using a NN. These models propose ways to perform simulations using data-driven physics, but none have, to the best of our knowledge, yet been applied to the problem of simulating packing pattern stability. Applying a data-driven physics model to the assessment of air cargo stability can speed up the optimization pipeline and allow for more computational resources to be allocated to finding better packing patterns with respect to utilized volume in short time windows.

The goal of this study is to illustrate the design and demonstration of a prototype data-driven model that predicts the stability of packing patterns in the context of air cargo palletizing. This work considers static stability, but a similar approach can also be applied to dynamic stability. On a conceptual level, this work treats the costly interplay between physical validation of interim solutions in a GA's fitness function and its speed-up using ML.

After presenting related work, we provide a description of our approach, which is based on the cross-industry standard process for data mining (CRISP-DM) framework, a common iterative methodology for data science projects that consists of six phases. Furthermore, we present our results, followed by a discussion, and conclusion of this work.

2 Theoretical Background

2.1 Stability Constraints in Air Cargo

Stability is imposed to preserve the (partly) stacked cargo on ULDs from collapsing [15]. A layout is statically stable, if the loaded items can "withstand the gravity force acceleration over them" [16], considering situations in which the cargo is loaded, and gravitational acceleration is applied. Dynamic stability describes the stability of cargo, in which the ULD is being moved the way it might be during transportation (e.g., during a flight). Both constraints originate from physical conditions, which are common for many practical optimization problems [17].

Among other approaches, physical simulations provide a realistic approximation of a packing pattern's stability. Especially the highly complex case of air cargo, with on the one hand high item heterogeneity [18] and on the other hand strict aviation safety regulations [5], requires accurate stability estimations. Moreover, desired high loading factors and strict time windows are requirements faced in practical operations [11]. Compared to other approaches, physical simulations ensure enough precision to get realistic physical feedback, as physical simulations build up a virtual image with physical laws similar to those in a real-world context (e.g. gravity, friction, collisions), providing sufficient interfaces and runtime performance to be used in heuristic search processes. The core of physics engines are dynamic simulations, which represent a physical system within a mathematical model, usually ordinary differential equations, and numerically solve it in a computer program [19]. Middleware applications that integrate dynamic simulations into other applications are called physics engines [8]. Recent articles demonstrate successful applications of physical simulations for static stability assessments in pallet loading contexts [6, 11].

However, physical simulations are costly to employ and require substantial amounts of computing power. As to be expected, increased quality of the computed solutions and considering more realistic physical assessments come at the expense of bad runtime performance [11]. Especially when embedded in time-sensitive applications such as games or optimization pipelines (e.g., heuristic search processes), this negatively impacts overall systems performance, since the simulation is executed for every potential solution candidate. In the case of optimizations, a long fitness evaluation per solution candidate results in longer overall runtimes per generation, resulting in fewer generations for the same time frame, and ultimately, a lower overall solution quality [6, 11]. For GA in general, "minimizing the number of fitness function calls is very important, if a call is expensive" [17].

Approaches of physical simulations within both PLP and CLP contexts can be classified into the categories (1) stand-alone and (2) integrated. The first category describes approaches that utilize physics engines to train or develop a model, which is deployed in an optimization algorithm. The latter represents approaches that directly call the simulation from within an optimization pipeline.

Examples for stand-alone approaches can be found in Martínez et al. [20], Ramos et al. [21], and Ramos et al. [7]. Approximations of the physical outcome are primarily captured in linear or mechanical models [20]. The authors use physics engines for verification purposes. The research of Ramos et al. [7] resulted in a stand-alone simulation software prototype (StableCargo) for evaluating the dynamic stability of container cargo. They benchmarked their physics engine-based standalone tool against a high precision engineering simulator and analytical solutions. They modelled different forces and velocities, which represent movements during transportation. Their tool provides an analytical approximation and high precision solutions. However, no report was made about the integration of their tool into an optimization pipeline.

In the study by Ramos et al. [21], a linear regression on dynamic stability scores, which were evaluated using physics engines, revealed a number of metrics and two performance indicators.

To the best of our knowledge, no works exist that employ ML-based physics prediction for the problem of static stability assessments.

2.2 Data-driven physics

In the context of this article, "data-driven physics" refers to the simulation of a physical system using models that learn the transition function between physical states from data of actual transitions. Predictions require fewer computations than the equivalent computations in a regular physics engine, resulting in a reduction in runtime [13].

Ajay et al. [22] present a variety of possible ways in which learned models can be integrated with analytical models (such as regular physics engines) to achieve different goals, including a reduction in the simulation runtime. The authors distinguished between "data-driven models" and "data-augmented models". Data-driven models replace regular physics engines by learning the transition function whereas data-augmented models use learned models on top of analytical physics engines to minimize the discrepancy between the simulation and the real-world. Holden et al. [13] projected a physical state (a vector of all of an object's vertex positions) into a lower-dimensional linear space using principal component analysis (PCA) and then perform the simulations in this subspace using a learned model. Any trained model of this kind is restricted to the physical objects it was trained on. It can thus be regarded as a trade-off between flexibility, i.e., it can deal with any kind of object, and speed.

Fulton et al. [23] also made use of PCA and a non-linear autoencoder to model the deformation of non-rigid bodies. Importantly, simulations of deformations are run in a low-dimensional non-linear latent space, which allows for quite complicated deformation dynamics to be learned. The simulation from one latent state to the next is not modeled using a NN but is instead performed by explicitly solving the equations of motion within this learned latent space. Wiewel et al. [14] used a convolutional NN and a long short-term memory (LSTM) network to encode a sequence of past states of a physical system of fluids to predict the next latent state, which, at the end of the sequence, can then be decoded into the next physical state. A predicted sequence of latent states can also be used to predict subsequent latent states.

The general idea of Wiewel et al. [14] is similar to that of Holden et al. [13] in that the learned transition function is applied to lower-dimensional, abstract states rather than concrete states describing the actual physical positions of items the way regular physics engines do. All data-driven physics models proposed ways of learning to simulate physical systems, but none have, to the best of our knowledge, yet been applied to the problem of simulating air cargo stability within optimization pipelines using GA, which is the focus of this study.

3 Methodology & Outcome

3.1 Approach

This research builds on the previous research by Lee et al. [5, 9] and Mazur et al. [11, 24] and utilizes the packing patterns calculated by the GA-based optimization pipeline [5], which considers numerous constraints, as well as the derived and defined metrics for stability calculation using physical simulations [11]. The ML approach used here can be seen as a significant enhancement of the existing physical simulation approach of Mazur et al. [11] as an integrative part of the GA-based optimization pipeline for stability assessment of generated packing patterns. As benchmark instances, we use the synthetic but realistic datasets of Brandt and Nickel [18], which are, to the best of our knowledge, both the first and only benchmark instances for air cargo palletization.

For our research we followed the CRISP-DM framework [25], which consists of six phases: (1) business understanding, (2) data understanding, (3) data preparation, (4) modeling, (5) evaluation and (6) deployment. The phase of data preparation included data generation and collection by having the GA propose packing patterns and simulating them in the *PyBullet* physics engine [26], which, in turn, yields the simulation data on which a data-driven model can be trained. Most of the data preparation, generation and modeling were conducted in the *Jupyter Lab* computational environment; primarily using the software library *pandas*. All models discussed in this work were implemented with *TensorFlow* (specifically *TensorFlow.Keras's* subclassing API) using the *Python* programming language. Deployment can also include the integration of the model into the existing system. Because of our goal to develop a prototype, a deployment is not part of the results.

3.2 Business Understanding

The underlying business domain of the addressed problem is air cargo palletizing, a highly constraint and complex type of both PLP and CLP [18]. Because of high item heterogeneity with respect to shapes and sizes, physics engines are employed to evaluate stability. Physics engines can deal with any kind and (computationally feasible) number of objects in their domain (e.g. rigid bodies) [8, 26].

3.3 Data Understanding

We impose several assumptions about the cargo and ULDs: all packing items are rigid bodies, uniformly dense, cuboidal and do not contain dangerous or fragile goods that warrant special treatment. Each simulation consists of a single ULD. Each ULD in turn consists of a variable number of packing items (though the number of items remains the same over the course of a single simulation). The number of items for a given simulation depends on the specific packing pattern proposed by the utilized GA [5].

To model the simulation of packing items inside a ULD, appropriate features must be picked. Possible features include item positions over time, item orientations over time, item shapes, item masses and the ULD shape. An item's position is described by a vector (x, y, z) that points to the center of the item at the start of the simulation, i.e., the position as proposed by the GA before any physical simulations have been applied. The shape of an item is defined by a vector (w, h, d), corresponding to the half-extents of the box, i.e., the orthogonal distance from an item's faces to its center point. Half-extents are a common way to encode axis-aligned bounding boxes for collision detection in physics engines. This also simplifies the application of the dataset to both physics engines and machine learning models. A packing item's shape is assumed to be constant over the course of a simulation since items are rigid bodies, such that they cannot deform because of collisions, falling on the floor, or a heavy item weighing down on it. The models described here consider positional and shape features of items since those are most relevant to static stability. Packing item masses are not explicitly modeled and instead assumed to be proportional to the item's volume (uniform density).

Data encoding. Each example's features can be described by two vectors: a position vector p (which will also be referred to as the "state" of the physical system at some timestep) and a shape vector s. Both vectors belong to a packing pattern with n items thus consist of $3n$ entries each: 3 positional coordinates in the positions vector, and 3 dimensions in the shape vector for each item. The order of items is based on the packing order as proposed by the GA.

$$p = (x_1, y_1, z_1, x_2, y_2, z_2, \dots, x_n, y_n, z_n) \tag{1}$$

$$s = (w_1, h_1, d_1, w_2, h_2, d_2, \dots, w_n, h_n, d_n) \tag{2}$$

$$x = \mathrm{concatenate}(\mathrm{p}, \mathrm{s}) \tag{3}$$

where the y-entries of p refer to the spatial y-coordinates of the items and not the example's target value (stability label). The final vector x concatenates both features.

The distinction between position and shape can be removed by modeling all vertices of every single item rather than center positions and half-extents. Since each item is assumed to be cuboidal, each item has 8 vertices à 3 coordinates, resulting in 24 entries per item. For a complete packing pattern, the input vector contains $24n$ entries (where n is the number of items in that packing pattern). Rather than explicitly modeling this vector, a transformed, more abstract, lower-dimensional vector can be used (e.g., by applying a dimensionality reduction method such as PCA [13] or an autoencoder [23, 27]).

3.4 Data Generation and Preparation

The data needed to learn the behavior of physical systems can be obtained by executing a regular physics engine or scientific simulation tool and storing the item positions (and possible orientations) at each timestep, and information about the simulation's set-up such as the item shapes. The PyBullet physics engine was used to simulate the physical behavior of a packing pattern whose initial positions are proposed by the utilized GA-based optimization pipeline [5]. The target (stability label) for each simulation, i.e. the value that describes whether the simulation determined that the packing pattern inside a single ULD is stable or unstable, is described by the amount of Euclidean distance between initial and final item position and measures the item's spatial displacement [11]. If any distance is greater than a predetermined threshold, the pattern is classified as unstable (0), and stable (1) otherwise. A comparison of the current stability metrics in PLP and CLP, as well as the derivation and definition of the stability metrics used in this paper, can be found in Mazur et al. [11].

One problem of this setup was the varying length of the input vector, which depends on the number of packing items in the packing pattern. In a multilayer perceptron (feed forward NN), each input vector must have the same size. The approach used here is to set a maximum number of items, n_{max}, and always assume that each packing pattern has exactly this number of items. If a pattern has fewer packing items, the remaining, unused vector entries (i.e., the positional and shape coordinates corresponding to the "missing" items) are simply set to zero, representing dimensionless, massless items that do not affect other items' positions.

To stabilize training, we standardized the data before fitting the model such that each feature had a mean of 0 and standard deviation of 1. Other scaling methods (normalization, minmax scaling) were also tried, but standardization performed best. The data was subsequently partitioned into training (60%), validation (20%) and test sets (20%). The full dataset consisted of 42,784 examples (of which 25,670 have been designated training examples), each of which corresponded to a simulation. An average simulation from this dataset contained 4.34 items whose positions are tracked over 460.74 timesteps of 1/80 s each.

3.5 Modeling

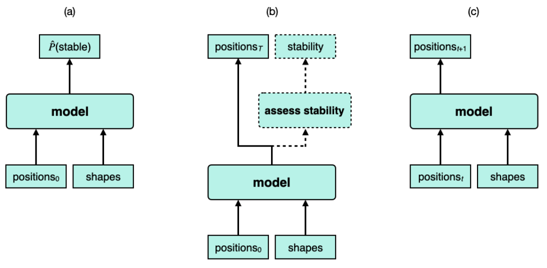

Figure 1. Approaches to modeling static stability

There are several ways to model stability predictions. The most straightforward approach is to directly map the packing pattern's stability label, i.e., to classify a pattern as either stable (1) or unstable (0) ("classification", as demonstrated in Figure 1a). Alternatively, the initial positions can be mapped to the final state of the physical system (“regression of the final state”) to which the same stability assessment (*assess_stability*) is then applied as in the case of using a regular physics engine (as demonstrated in Figure 1b). Finally, the positions at any given timestep t and the packing item shapes can be mapped to the positions at the subsequent timestep $t+1$ (“regression of the next state”, as demonstrated in Figure 1c). The last model comes closest to what a regular physics engine would do by modeling intermediate steps. All models are depicted in Figure 1. In the following, each model is described in more detailed.

Stability classification. Model (a) requires a dataset of initial positions, shapes and the stability label of the corresponding physical system as determined by the physics engine. As baseline, we fitted a logistic regression (LR) model to the data. LR applies the logistic function to an affine transformation of the input features, thereby mapping each example to a value between 0 and 1, which can be interpreted as the predicted probability of the example being statically stable: $\hat{y}^{(\mathrm{i})} = \hat{P}(\text{"example } (i) \text{ is statically stable"})$. Then, a NN was fitted to the same data with the goal of improving upon this baseline accuracy. The number of units per layer (which depends on the input vector size and therefore on n_{max}), the number of layers, and which activation functions to use were gradually increased to find the architecture that yielded the highest validation accuracy. Ultimately, the exponential linear unit (ELU) activation function for all layers except for the last worked best. The last layer is a sigmoid function mapping the output of an affine transformation to the interval $\{0,1\}$ that corresponds to the probability of the pattern being stable. The final sigmoid layer can be interpreted as a LR classifier that is applied, not to the raw input features, but to learned, non-linear features returned by the previous NN layers [12].

Final state regression. Regression is a different approach to solving this problem. This model predicts final spatial positions of the bodies in the simulation, i.e., the state in which the positions no longer or only negligibly change. It performs the computation that provide an approximation of actual physical simulation of the system, but without explicitly modeling the intermediate steps. Final positions can be passed to stability assessment (*assess_stability*), which then returns the stability label. From these predicted stability labels and the actual stability labels (the targets provided by the physics engine), the accuracy can be computed and compared to those of the classification models. Varying packing patterns can require different numbers of timesteps to reach the final state from the initial state. In the following, $T(i)$ denotes the timestep of the i-th example’s final state (where the initial state has timestep 0) but is abbreviated to T for all examples for simplicity. The total number of transitions in this sequence is T, and the total number of states is $T+1$. Furthermore, rather than directly predicting the full final state p_T, the regression model is trained to predict the difference vector to the initial state δ, where $\hat{\delta} = \mathrm{f}(\mathrm{p}_0, \mathrm{s})$:

$$\hat{p}_T = \mathrm{p}_0 + \hat{\delta} \tag{4}$$

Next state regression (transition model). The second regression model attempts to learn transition functions between states. The inputs are the positions and shapes at some timestep t, which are then mapped to the positions at the subsequent timestep $t +$

1. This "next state regression" can be applied to each newly predicted state until the final state (when positions no longer or only minimally change) is reached (see Figure 2). The regression models predict the next state in an iterative (right, also see Figure 1c) or recurrent (left) fashion. The architecture of the "model" module and the shape vector are the same across all timesteps. In the recurrent model, a "hidden state" is transmitted from one timestep to the next.

This approach makes use of the model as a transition function between physical states to predict the full sequence of future states. As with model (b), this model does not predict the full state, but just the difference between the input positions and target positions, δ_{t+1}:

$$\hat{p}_{t+1} = \mathrm{p}_t + \hat{\delta}_{t+1} \tag{5}$$

where $\hat{\delta}_{t+1} = \mathrm{f}(\mathrm{p}_t, s)$. Holden et al. [13] and Wiewel et al. [14] propose that training not on single transitions but rather on a "window" of transitions $\{(p_t, p_{t+1}), (p_{t+1}, p_{t+2}), \dots, (p_{t+w-1}, p_{t+w})\}$ for some window size w, or even the full simulation sequence, makes training more effective and stable. A sequence of state vectors constitutes a trajectory through state space. Using a loss function ensures that not only predicted single-timestep transitions are close to target transitions but that the trajectory is close to the target trajectory.

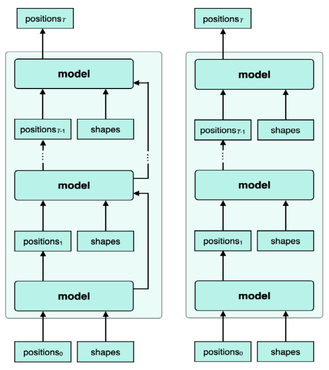

Figure 2. Recurrent (left) and iterative (right) regression model

A recurrent NN could achieve greater accuracy by allowing information to be passed across timesteps and thereby providing information about the past trajectory, which in turn informs the future trajectory. Recurrent NNs are frequently applied to sequential data [12] and could therefore be an appropriate model to address the problem of simulating physical states over a sequence of timesteps [13, 14]. A recurrent architecture where the output from the previous timestep, $t-1$, is fed into the model as the input of the current timestep, t, is commonly used in the natural language processing domain [28]. In this setting, a decoder network maps a "thought vector" that encodes the information of an input sequence (e.g., a sentence in language A) to an output sequence (e.g. a sentence in language B). A similar architecture is used in the data-driven physics domain by Navrátil et al. [29]. Similarly, in this approach, a concrete packing pattern vector p_t is mapped to the corresponding, subsequent simulation sequence $(p_{t+1}, p_{t+2}, \ldots, p_T)$.

As Ajay et al. [22] point out, a recurrent architecture can also help alleviate the problem of compounding errors. As soon as the "next state regression" model has predicted a new state, this new state can then be used to predict the subsequent state. On the other hand, errors occurring at any point in the simulation will accumulate, possibly resulting in a wrong final state. There is no mechanism to ensure that not just each individual transition is as close to the real transition, but that the trajectory is also as close to the full target trajectory. Computing the loss over a window of transitions or even the full sequence and adding recurrent connections between timesteps can help soften this problem.

All recurrent NNs implemented in this project are based on the LSTM network, which is generally better at storing information over long timespans by selectively remembering and forgetting old information (e.g. from previous timesteps) and keeping and discarding new information (e.g. from the current timestep) [30]. The iterative "next state regression" models implemented in this project only consider a fixed number of 5 time units. The initial state always happens at time unit 0 and the final state at time unit 4. The intermediate states are evenly spaced between them. Table 1 provides an overview about the models presented.

Table 1. NN models

Model	**Activation**	**Activation (last layer)**	**Input**	**Target**	**Loss function**
Classifier	ELU	sigmoid	(p_0, s)	y	BCE
Classifier	ELU	sigmoid	(p_0, s)	y	BCE
Classifier	ELU	sigmoid	(p_0, s)	y	BCE
Final state regressor	ELU	none	(p_0, s)	p_T	MSE
Next state regressor	ELU	none	(p_t, s)	p_{t+1}	MSE

3.6 Evaluation

To evaluate the runtime of the physics engine and models, 10,000 simulations were randomly sampled from those containing no more than n_{max} packing items. For each $n_{max} \in \{3, 8, 12\}$, all 10,000 simulations were run $n_{runs} = 5$ times for the physics engine and n_{runs}=1,000 times for the NNs and LR. The mean and standard deviation of the runtime (in ms) are reported in Table 2 ($n_{max} = 8$) and Table 3 ($n_{max} = 12$).

From among those models for which data is available, the NN classifier achieves the highest accuracy for all n_{max}. Due to its very small size, the LR model, which for the purpose of this study acts as benchmark, is by far the fastest. However, the NN also demonstrates a significant speed-up over the physics engine, which performs worst in terms of runtime. The physics engine achieves an accuracy of 100% since it generated the training targets. The trade-off between accuracy and runtime can inform a decision about which configuration is most appropriate for a given scenario.

The column n_{runs} refers to the number of times all 10,000 simulations were run to evaluate the mean and standard deviation of the simulation (or prediction) runtime. Accuracy was computed on the test dataset on which none of the models had been trained.

Table 2. Runtime and performance evaluation for $n_{max} = 8$.

$n_{max} = 8$				Runtime per simulation (ms)	
Model	n_{runs}	**#Params.**	**Accuracy**	**Mean**	**Std. dev.**
Physics Engine	5	-	1	$1.23 * 10^2$	$3.00 * 10^0$
Classifier (LR)	1,000	49	0.7975	$4.56 * 10^{-10}$	$1.50 * 10^{-10}$
Classifier (NN)	1,000	5,953	0.9051	$4.04 * 10^{-2}$	$1.77 * 10^{-2}$

Table 3. Runtime and performance evaluation for $n_{max} = 12$.

$n_{max} = 12$				Runtime per simulation (ms)	
Model	n_{runs}	**#Params.**	**Accuracy**	**Mean**	**Std. dev.**
Physics Engine	5	-	1	$1.27 * 10^2$	$7.87 * 10^0$
Classifier (LR)	1,000	73	0.8129	$5.76 * 10^{-10}$	$3.05 * 10^{-10}$
Classifier (NN)	1,000	5,329	0.9054	$2.56 * 10^{-2}$	$4.10 * 10^{-3}$

4 Discussion and Future Research

The developed models provide a proof-of-concept that a data-driven physics model can predict air cargo static stability labels obtained by a physics engine with around 90% accuracy, doing so many orders of magnitudes faster than a regular physics engine. This alleviates bottlenecks in the packing pattern optimization pipeline and allows more computational resources to be allocated to the search for better packing patterns by the applied GA [5, 6, 11] when the ML approach is integrated into the optimization system.

Despite its satisfying accuracy, the approach should only be considered in relation to the physics engine, which is itself an approximated representation of reality. Such a layered abstraction can lead to an additive reduction in accuracy compared to reality. The consequence would be wrong predictions in the stability assessment, so that practical constraints are no longer met and aviation safety may be negatively affected.

However, predicting air cargo stability instead of running the numerical simulation of a physics engine can be a starting point for further studies and improvements. A combination of both physics engine and data-driven model can provide a trade-off between speed and accuracy, which, in conclusion impacts aviation safety, that might be better than either one alone. Since a simulation must be run for each loading step of the packing pattern, most simulations concern partially loaded patterns, which can be replaced by stability predictions. To ensure that only those packing patterns are accepted that are statically stable, the system can then switch over to the physics engine to validate the static stability of the full packing pattern. Furthermore, the stability of each loading step of the final packing pattern in the last iteration can also be validated to guarantee correctness (at least with respect to the correctness of the physics engine).

The work presented in this study is a proof-of-concept and contains several limitations. Firstly, even though it was suggested above to use a data-driven physics model for partially loaded packing patterns to reduce the overall runtime of the optimization pipeline, the models described in this study were only trained on full packing patterns (though, ranging from a scope of 1 to 17 packing items per pattern). Since the implemented models can deal with packing patterns of a wide range of numbers of packing items, a model is likely to deal with partially loaded packing patterns. Further, all simulation data and classification targets used for this work are based on data generated by a physics engine. Physics engines are not, however, perfect reflections of the real world. To obtain higher-quality data, the simulation data could instead be generated by high-performance scientific simulators, which have greater accuracy than physics engines, as demonstrated in the context of air cargo load planning by Martinez-Franco and Álvarez-Martínez [8] or directly from the real world data. Finally, the masked representation of packing patterns with fewer items might lead to lower accuracy than is theoretically achievable and puts an upper limit to the number of items the model can handle.

In future research, completely different approaches can be explored, such as graph NNs, relational representations between items, or probabilistically modeling the transition function the way it is commonly done in reinforcement learning to model the environment's dynamics [31].

5 Conclusion

This study presents how to design a data-driven model that can predict the static stability of air cargo packing patterns with significant speed-up over regular physics engines and with enough accuracy to take a large step closer towards operational value. The speed-up can also have a positive effect on time savings in the operational processes of air cargo palletizing, which is characterized by enormous time pressure, especially in the area of transfer and build-up of the packages [18]. Near-time calculation of packing patterns with packages already on site for physical build-up is the first step towards the automation of an area that is currently heavily dominated by manual processes [5, 9]. This work also presents ways in which to combine a data-driven physics model with a regular physics engine to take advantage of the best properties of both approaches. We present an approach to cope with the runtime-intensive complexity of physical validation during evolutionary optimization processes and contribute to the body of knowledge on how to employ data-driven ML techniques and GAs from a practical viewpoint. Finally, this work proposes a variety of routes of possible future research on how to improve upon the model to further increase accuracy and reduce runtime, such as alternate ways of encoding the dataset, representing the input vector, generating a higher-quality dataset, augmenting the dataset, and laying out a more sophisticated NN architecture.

References

1. Bortfeldt, A., Wäscher, G.: Constraints in container loading-A state-of-the-art review. In: European Journal of Operational Research. pp. 1–20. Elsevier B.V. (2013). https://doi.org/10.1016/j.ejor.2012.12.006.
2. Zhao, X., Bennell, J.A., Bektaş, T., Dowsland, K.: A comparative review of 3D container loading algorithms. International Transactions in Operational Research. 23, 287–320 (2016). https://doi.org/10.1111/itor.12094.
3. Gehring, H., Bortfeldt, A.: A genetic algorithm for solving the container loading problem. International Transactions in Operational Research. 4, 401–418 (1997). https://doi.org/10.1016/S0969-6016(97)00033-6.
4. Gonçalves, J.F., Resende, M.G.C.: A parallel multi-population biased random-key genetic algorithm for a container loading problem. Computers & Operations Research. 39, 179–190 (2012). https://doi.org/10.1016/j.cor.2011.03.009.
5. Lee, N.-S., Mazur, P.G., Bittner, M., Schoder, D.: An Intelligent Decision Support System for Air Cargo Palletizing. In: Proceedings of the 54th Hawaii International Conference on System Sciences, January 5-8, 2021, Hawaii, USA (2021). https://doi.org/10.24251/HICSS.2021.170.
6. Bracht, E.C., de Queiroz, T.A., Schouery, R.C.S., Miyazawa, F.K.: Dynamic cargo stability in loading and transportation of containers. In: 2016 IEEE International Conference on Automation Science and Engineering (CASE). pp. 227–232. IEEE (2016). https://doi.org/10.1109/COASE.2016.7743385.
7. Galrão Ramos, A., Jacob, J., Justo, J.F., Oliveira, J.F., Rodrigues, R., Gomes, A.M.: Cargo dynamic stability in the container loading problem - a physics simulation tool approach. International Journal of Simulation and Process Modelling. 12, 29–29 (2017). https://doi.org/10.1504/IJSPM.2017.082788.
8. Martinez-Franco, J.C., Alvarez-Martinez, D.: Physx as a middleware for dynamic simulations in the container loading problem. In: 2018 Winter Simulation Conference (WSC). pp. 2933–2940. IEEE (2018). https://doi.org/10.1109/WSC.2018.8632469.
9. Lee, N.-S., Mazur, P.G., Hovestadt, C., Schoder, D.: Designing a State-of-The-Art Information System for Air Cargo Palletizing. In: 15th International Conference on Design Science Research in Information Systems and Technology, DESRIST 2020, Kristiansand, Norway, December 2–4, 2020, Proceedings. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-64823-7_36.
10. Krebs, C., Ehmke, J.F.: Axle Weights in combined Vehicle Routing and Container Loading Problems. EURO Journal on Transportation and Logistics. 10, 100043 (2021). https://doi.org/10.1016/j.ejtl.2021.100043.
11. Mazur, P.G., Lee, N.-S., Schoder, D.: Integration of Physical Simulations in Static Stability Assessments for Pallet Loading in Air Cargo. In: Bae, K.-H., Feng, B., Kim, S., Lazarova-Molnar, S., Zheng, Z., Roeder, T., and Thiesing, R. (eds.) Proceedings of the 2020 Winter Simulation Conference, December 13-16, Orlando, Florida, USA (2020). https://doi.org/10.1109/WSC48552.2020.9383878.
12. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. The MIT Press, Cambridge, Massachusetts (2016).
13. Holden, D., Duong, B.C., Datta, S., Nowrouzezahrai, D.: Subspace neural physics: fast data-driven interactive simulation. In: Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 1–12. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3309486.3340245.

14. Wiewel, S., Becher, M., Thuerey, N.: Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow. Computer Graphics Forum. 38, 71–82 (2019). https://doi.org/10.1111/cgf.13620.
15. Bischoff, E.E.: Stability aspects of pallet loading. OR Spektrum. 13, 189–197 (1991). https://doi.org/10.1007/BF01719394.
16. Junqueira, L., Morabito, R., Sato Yamashita, D.: Three-dimensional container loading models with cargo stability and load bearing constraints. Computers and Operations Research. 39, 74–85 (2012). https://doi.org/10.1016/j.cor.2010.07.017.
17. Kramer, O.: Genetic Algorithm Essentials. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-52156-5.
18. Brandt, F., Nickel, S.: The air cargo load planning problem - a consolidated problem definition and literature review on related problems. European Journal of Operational Research. 275, 399–410 (2019). https://doi.org/10.1016/j.ejor.2018.07.013.
19. Witkin, A., Baraff, D.: An Introduction to Physically Based Modeling: Differential Equation Basics. SIGGRAPH Course Notes 1997. (1997).
20. Martínez, J.C., Cuellar, D., Álvarez-Martínez, D.: Review of Dynamic Stability Metrics and a Mechanical Model Integrated with Open Source Tools for the Container Loading Problem. Electronic Notes in Discrete Mathematics. 69, 325–332 (2018). https://doi.org/10.1016/j.endm.2018.07.042.
21. Galrão Ramos, A., Oliveira, J.F., Gonçalves, J.F., Lopes, M.P.: Dynamic stability metrics for the container loading problem. Transportation Research Part C: Emerging Technologies. 60, 480–497 (2015). https://doi.org/10.1016/j.trc.2015.09.012.
22. Ajay, A., Wu, J., Fazeli, N., Bauza, M., Kaelbling, L.P., Tenenbaum, J.B., Rodriguez, A.: Augmenting Physical Simulators with Stochastic Neural Networks: Case Study of Planar Pushing and Bouncing. arXiv:1808.03246 [cs]. (2018).
23. Fulton, L., Modi, V., Duvenaud, D., Levin, D.I.W., Jacobson, A.: Latent-space Dynamics for Reduced Deformable Simulation. Comput. Graph. Forum. (2019). https://doi.org/10.1111/cgf.13645.
24. Mazur, P.G., Lee, N.-S., Schoder, D., Janssen, T.: Designing a Physical Packing Sequence Algorithm with Static Stability for Pallet Loading Problems in Air Cargo State of the Art. In: 12th International Conference on Computational Logistics, ICCL 2021, Enschede, The Netherlands, September 27–29, 2021, Proceedings (2021).
25. Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R.: CRISP-DM 1.0: Step-by-step data mining guide. undefined. (2000).
26. Coumans, E., Bai, Y.: Pybullet, a python module for physics simulation in robotics, games and machine learning. (2017).
27. Hinton, G.E., Salakhutdinov, R.R.: Reducing the Dimensionality of Data with Neural Networks. Science. 313, 504–507 (2006). https://doi.org/10.1126/science.1127647.
28. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to Sequence Learning with Neural Networks. In: Advances in Neural Information Processing Systems. Curran Associates, Inc. (2014).
29. Navrátil, J., King, A., Rios, J., Kollias, G., Torrado, R., Codas, A.: Accelerating Physics-Based Simulations Using End-to-End Neural Network Proxies: An Application in Oil Reservoir Modeling. Frontiers in Big Data. 2, 33 (2019). https://doi.org/10.3389/fdata.2019.00033.
30. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Comput. 9, 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735.

31. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. arXiv:1602.01783 [cs]. (2016).