DAG Card is the new Model Card

Jacopo Tagliabue[*]
Coveo Labs
jtagliabue@coveo.com

Ville Tuulos
Outerbounds
ville@outerbounds.co

Ciro Greco
Coveo Labs
cgreco@coveo.com

Valay Dave
Outerbounds
valay@outerbounds.co

Abstract

With the progressive commoditization of modeling capabilities, data-centric AI recognizes that what happens before and after training becomes crucial for real-world deployments. Following the intuition behind *Model Cards*, we propose *DAG Cards* as a form of documentation encompassing the tenets of a data-centric point of view. We argue that Machine Learning pipelines (rather than models) are the most appropriate level of documentation for many practical use cases, and we share with the community an open implementation to generate cards from code.

1   Introduction

While software is eating the world, the growing use of machine learning (ML) in production systems makes it increasingly hard to understand exactly *how*. Even setting aside interpretability issues [12], ML models are harder to test and debug compared to traditional software. To this extent, Mitchell et al. [13] introduced *Model Cards*, i.e. short documents accompanying models that provide information on accuracy, biases, limitations and best practices for their use. Model Cards are meant to provide a "reference for all, regardless of expertise" [1]: "ML folks" may check details on architecture and metrics, "product folks" get to know strengths and weaknesses of the model in different scenarios, while final users – when applicable – can understand better how model decisions impact their lives [19].

Recently, proponents of data-centric AI [15, 9] raised awareness about the importance of what surrounds model choices as well, arguing that what happens before and after training is as important as modelling [11]. Take the latter point at face value, we believe that documentation should incorporate those aspects as well. In *this* work, we motivate the move from Model Cards to *Directed Acyclic Graph (DAG) Cards*, and share with the community an implementation for *Metaflow* [4], the popular framework for ML workflows.

2   From models to pipelines

Data-centric AI sits at the conjunction of two theses: on one side, modelling has been increasingly commoditized [14], as deep learning's inherent flexibility and pre-trained models provide good out-of-the-box performances; on the other, the Cambrian explosion [10] of MLOps tools brought more attention on the fundamental importance of what happens before (data collection, preparation, labeling, quality [17]) and after (testing, serving, monitoring, drifting) the modelling stage [18, 20].

---

[*]Jacopo designed the first card prototype, which kick-started the project; Ville created Metaflow while working at Netflix and he is one of its main maintainer; Ciro worked on the paper draft and researched the context for the article; finally, Valay implemented cards in Metaflow.

While the original cards [1] showcased a B2C scenario, where APIs are public and use cases require no specific knowledge, Fig. 1 shows a more typical setting for ML developers: we borrow the pipeline from Chia et al. [8] – a recent research paper built with Metaflow – to have a specific example in mind, but the structure is fairly common to many modern pipelines [22]. As any practitioner knows, "training a model" involves stitching together a heterogeneous set of functionalities, from gathering data to cleaning / aggregating it, from preparing features to evaluating performances. In other words, "training a model" is not an atomic operation, but a series of tasks with explicit dependencies: some tasks need to be executed before others, some can run in parallel; failures may require restarting some tasks, but not all of them – and so on.

Conceptually, we move from the typical research setting of stand-alone scripts, to the more structured concept of DAG: while dependency management was present in earlier frameworks as well [5], Metaflow goes one step further, and provides the possibility of picking *per task* a particular combination of Python packages and computational resources.
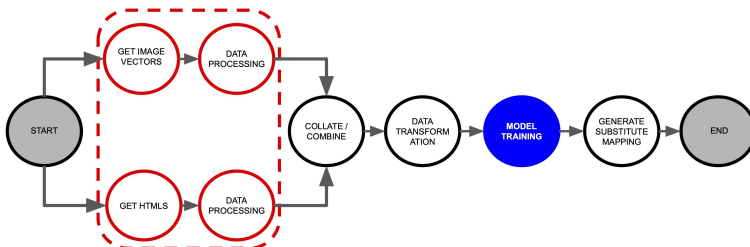

Figure 1: DAG for the e-commerce substitute model proposed by Chia et al. [8]. Tasks in *red* are run in parallel, tasks in *blue* are selectively executed on GPU-accelerated hardware.

Data-centric AI understands that all tasks – not just modelling – play a crucial role in determining the final behavior of the system, when deployed: if complex software systems have been long evaluated through "behavioral testing" [3], Ribeiro et al. [16] argue that it is time for ML to do the same [7]. In other words, given that data quality, feature preparation and choice of metrics influence the output of the system in the wild, properly documenting models involves documenting the entire ML pipeline.

3 Modern ML pipelines with Metaflow

Metaflow was originally developed at Netflix and it is now available as an open source package at `https://github.com/Netflix/metaflow`. By providing a Python-first, local-to-cloud integrated environment, Metaflow nudges users into following data-centric best practices through its design. In particular, Metaflow provides three key features to support reproducible and robust ML pipelines:

1. **workflow definition through DAG**: by structuring the application as a DAG-based workflow, it replaces sets of *ad hoc* scripts and unreliable piecemeal execution. Furthermore, Metaflow introduces a shared lexicon for the team to model and discuss pipelines in a consistent way: *Flows* (the DAG), *Steps* (nodes of the graph), *Tasks* (units of execution yielded by steps), and *Data artifacts* (the task state). When a flow is executed either manually or by a scheduler, the execution is tracked as a *Run*.

2. **Reproducible, isolated execution of each step**: Metaflow packages together user-defined code and 3rd party dependencies in an immutable code package, which can be shipped and executed on various compute layers. Variables (such as data and states) are also automatically stored (see below).

3. **Abstraction of computation and scheduling**: Metaflow comes with built-in integrations to popular cloud-based batch compute layers, such as AWS Batch or Kubernetes, and schedulers, like AWS Step Functions.

Moreover, Metaflow and data-centric AI share the core assumption that the final behavior of ML systems is due to the interaction of two equally important components: modelling *and* data. As such, Metaflow ships with additional features that make it an ideal tool to support automated generation of DAG Cards:

2

1. **High-throughput access to large datasets** - ML workflows are data-intensive by nature and it is common for functions to be IO-bound. Metaflow provides tools and patterns for quickly moving large datasets from a cloud-storage directly to the process memory.

2. **Snapshotting and versioning of all artifacts**: code, results and execution metadata are automatically tracked and persisted in an immutable data store [2]; dataset, features, model weights can be stored and versioned as well using built-in functionalities. While the importance of version control for the software/modelling aspect of a pipeline was recognized a long time ago, Metaflow emphasizes recording and tracking all inputs, intermediate states and data features as well, for full reproducibility.

3. **Native support for experimentation**: if it is true that data-centric AI is a fundamentally iterative activity, it is crucial to prototype quickly, without sacrificing reproducibility and rigor. Metaflow supports artifact inspection and *ad hoc* analysis through notebooks, and it can accommodate the use of any off-the-shelf library for data analysis, data QA and modelling, ensuring that data scientists are able to choose the best approach for each use case.
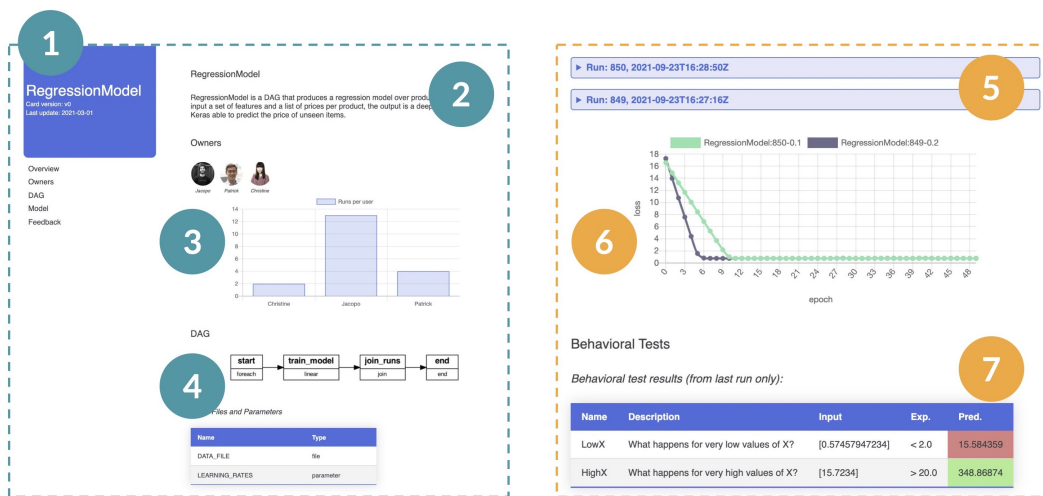
4 DAG Cards

Figure 2: Sample of DAG Card rendering from the code by Tagliabue [21]: on the left, Flow-level information (green); on the right, Run-level data (orange). Note that all details are automatically generated by comments, code introspection and third-party APIs, with no manual intervention.

We designed *DAG Cards* as a team of practitioners leveraging our extensive combined experience in ML for e-commerce, customer service and the entertainment industry. It is easy to realize the advantages of DAG Cards, especially when compared to popular tools of knowledge management that are ubiquitous in the industry: on the one hand, traditional software documentation - such as API docs -, on the other, internal product memos - such as intranet wiki-like pages. Based on our experience transitioning from these approaches to cards, we wish to highlight the following usability and philosophical principles:

1. **low effort**: cards require a small effort from the developer – as long as the code is well-commented/decorated, changing the code automatically results in new, up-to-date documentation. Embracing the "documentation-as-code" idea has two intended consequences: on the organizational side, we don't promote the hand-over of explaining a system from engineers to PMs, but instead require ML developers to be responsible for (at least some of) the explanatory artifacts; on the practical side, we remove the additional step – often neglected, postponed, forgotten – of updating some *other* website after pushing working code; every time this step is forgotten, documentation gets stale, less useful and in some cases even potentially harmful;

2. **versioning**: cards are artifacts themselves and can be generated after each run with pointers to the dataset, model weights, and all other artifacts in use; every card can be stored as an immutable, self-sufficient record of a particular run; as such, practitioners are able to quickly know how to debug specific states or even re-run the Flow entirely;

3. **extensibility**: cards can be extended to incorporate all sorts of information (including interactive elements) and, more generally, act as a one-stop repository for data stored in multiple tools (e.g. retrieve accuracy from an experiment tracking system). Not only internal wiki pages are static, but they often paint an incomplete picture, as it is not obvious that authors have access to all the third-party tools storing the required contextual data.

To make things more concrete, Fig. 2 displays some selected features of a sample DAG card, as a vanilla web page so it would be easy to store and to share through a web browser. Flow-level data are in *green*, Run-level data in *orange*:

1. **Title and menu**: the name of the Flow implicitly names the card and provides quick access to the card sections;

2. **DAG description**: the section describes what the pipeline is for and provides contextual information; it is rendered automatically through Python introspection, i.e. using `obj._doc_` to retrieve the class docstring.

3. **Ownership**: the section provides information about the users working on the DAG and plots the distribution of the runs to get a sense of the relative involvement of the team members.

4. **Structure and parameters**: the section prints out the DAG for easy visual inspections of the dependencies and tracks the parameters and input files determining the pipeline behavior; files are automatically tracked by Metaflow and can easily be linked to the DAG card for speedy lookup.

5. **Model architecture and training info**: the section reports for the last k (here $k = 2$) runs specific info about training (e.g. training and validation loss, as recorded by a model-independent third-party tool [6]) and a serialized representation of the chosen architecture (e.g. the output of Keras `model.summary`).

6. **Loss (or accuracy) per epoch chart**: using data from a third-party tool, the section charts the loss/epoch chart for the last k runs.

7. **Behavioral tests**: ML systems should be stress-tested on input-output pairs of particular interest in a "black box" fashion [7, 16]. Behavioral tests can, for example, highlight performance deficits on a subset of users (in analogy with CATE in A/B testing [23]), check for regression errors, control for edge cases and act as sanity checks against known problematic inputs. Of all the sections, this is the one encouraging collaboration between engineering and product teams the most.

Following a stand-alone prototype [21], we are now releasing a first implementation of DAG Cards within the official *Metaflow* codebase. The same approach we champion can obviously be extended to other DAGs: by open-sourcing our implementation of the above principles, we hope to provide practical guidance to help extending cards to other tools in the community.

5   Conclusion

We argued that ML pipelines are the most appropriate level of analysis for ML systems, and surveyed what makes Metaflow suitable for data-centric AI workloads. Leveraging its abstractions, we propose a first open-source implementation of DAG Cards, and highlight important principles so that the same features could easily be ported to other tools.

If, as argued in Molino and Ré [14], the next wave of ML systems will truly allow a substantially larger number of people to train and deploy models, documenting *pipelines*, not just (commoditized) models, will be more and more important for all the stakeholders involved. DAG Cards are just a first step towards this future.

Acknowledgments and Disclosure of Funding

References

[1] Face detection, 2020. URL https://modelcards.withgoogle.com/face-detection.

[2] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8:1792–1803, 2015.

[3] B. Beizer and J. Wiley. Black box testing: Techniques for functional testing of software and systems. *IEEE Software*, 13(5):98–, 1996. doi: 10.1109/MS.1996.536464.

[4] D. Berg, R. K. Chirravuri, R. Cledat, S. Goyal, F. Hamad, and V. Tuulos. Open-sourcing metaflow, a human-centric framework for data science, 2019.

[5] E. Bernhardsson and E. Freider. Luigi. https://github.com/spotify/luigi, 2012.

[6] L. Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

[7] P. J. Chia, J. Tagliabue, F. Bianchi, C. He, and B. Ko. Beyond ndcg: behavioral testing of recommender systems with reclist. *ArXiv*, 2021.

[8] P. J. Chia, B. Yu, and J. Tagliabue. Are you sure?: Preliminary insights from scaling product comparisons to multiple shops. In *SIGIR eCom 2021*, 2021.

[9] K. Goel and L. Orr. Data-centric ai. https://github.com/HazyResearch/data-centric-ai, 2012.

[10] C. Huyen. Mlops tooling landscape, 2021. URL https://huyenchip.com/2020/12/30/mlops-v2.html.

[11] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

[12] Z. C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, June 2018. ISSN 1542-7730. doi: 10.1145/3236386.3241340. URL https://doi.org/10.1145/3236386.3241340.

[13] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, page 220–229, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361255. doi: 10.1145/3287560.3287596. URL https://doi.org/10.1145/3287560.3287596.

[14] P. Molino and C. Ré. Declarative machine learning systems: The future of machine learning will depend on it being in the hands of the rest of us. *Queue*, 19(3):46–76, June 2021. ISSN 1542-7730. doi: 10.1145/3475965.3479315. URL https://doi.org/10.1145/3475965.3479315.

[15] A. Ng. Ai doesn't have to be too complicated or expensive for your business, 2021. URL https://hbr.org/2021/07/ai-doesnt-have-to-be-too-complicated-or-expensive-for-your-business.

[16] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh. Beyond accuracy: Behavioral testing of nlp models with checklist. In *ACL*, 2020.

[17] A. Rogers. Changing the world by changing the data. In *ACL/IJCNLP*, 2021.

[18] N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P. K. Paritosh, and L. M. Aroyo. "everyone wants to do the model work, not the data work": Data cascades in high-stakes ai. 2021.

[19] C. Seifert, S. Scherzinger, and L. Wiese. Towards generating consumer labels for machine learning models. *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*, pages 173–179, 2019.

[20] J. Tagliabue. You do not need a bigger boat: Recommendations at reasonable scale in a (mostly) serverless and open stack. In *Fifteenth ACM Conference on Recommender Systems*, RecSys '21, page 598–600, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384582. doi: 10.1145/3460231.3474604. URL `https://doi.org/10.1145/3460231.3474604`.

[21] J. Tagliabue. Dag card. `https://github.com/jacopotagliabue/dag-card-is-the-new-model-card`, 2021.

[22] J. Tagliabue, P. J. Chia, A. Sutcliffe, and L. Bigon. You don't need a bigger boat. `https://github.com/jacopotagliabue/you-dont-need-a-bigger-boat`, 2021.

[23] J. Wong, R. A. Lewis, and M. Wardrop. Efficient computation of linear model treatment effects in an experimentation platform. *arXiv: Computation*, 2019.