To We Robot Participants:

Thank you for reading! Despite the formatting, this is still definitely a draft, as you'll be able to tell once you get to the unfinished sections. Along with any other feedback, we're surely missing lots of citations, so if there's anything you think we should cite or read, please do not be shy in telling us that. We look forward to your comments and questions!

--Andrew, Suresh & Lizzie

THE LEGAL CONSTRUCTION OF BLACK BOXES

Andrew D. Selbst[*]
Suresh Venkatasubramanian[**]
I. Elizabeth Kumar[***]

Abstraction is a fundamental technique in computer science. Formal abstraction treats a system as defined entirely by its inputs, outputs, and the relationship that transforms inputs to outputs. If a system's user knows those details, they need not know anything else about how the system works; the internal elements can be hidden from them in a "black box." Abstraction also entails abstraction choices: What are the relevant inputs and outputs? What properties should the transformation between them have? What constitutes the "abstraction boundary?" These choices are necessary, but they have implications for legal proceedings that involve the use of machine learning (ML).

This paper makes two arguments about abstraction in ML and legal proceedings. The first is that abstraction choices that can be treated as normative and epistemic claims made by developers that compete with judgments properly belonging to courts. Abstraction constitutes a claim as to the division of responsibility: what is inside the black box is the province of the developer; what it outside belongs to the user. Abstraction also is a factual definition, rendering the system an intelligible and interrogable object. Yet the abstraction that defines the boundary of a system is itself a design choice. When courts treat technology as a black box with a fixed outer boundary, they are unwittingly abdicating their responsibility to make normative judgments as to the division of responsibility for certain wrongs, and abdicating part of their factfinding roles by taking the abstraction boundaries as a given. We demonstrate these effects in discussions of foreseeability in tort law, liability in discrimination law, and evidentiary burdens more broadly.

Our second argument builds from that observation. By interpreting the abstraction as one of many possible design choices, rather than a simple fact, courts can surface those

[*] Author's Note 1
[**] Author's Note 2
[***] Author's Note 3

choices as evidence to draw new lines of responsibility without necessarily interrogating the interior of the black box itself. Courts may draw on evidence about the system as presented to support these alternative lines of responsibility, but by analyzing the construction of the implied abstraction boundary of a system, they can also take the context around its development and deployment into account. Courts can rely on experts to compare a designer's choices with emerging standard practices in the field of ML or assign a burden to a user to justify their use of off-the-shelf technology. After resurfacing the normative and epistemic contentions embedded in the technology, courts can use familiar lines of reasoning to assign liability as proper.

INTRODUCTION

The "black box" is ubiquitous in narratives surrounding machine learning (ML), artificial intelligence (AI), and accountability. The black box is usually the villain of the story, the reason that there cannot be accountability for the harms associated with algorithmic systems. The black box is a trade secret.[1] It's inscrutable.[2] It's both.[3] But while most analyses focus on the blackness of the box—its opacity, its mystery—few examine the contours of the box itself.[4] This Article is about the construction of the box—its purpose, the choices that determine it, who gets to decide the proper boundaries of the system, and how this all affects liability.

When people are injured by algorithmic systems, whether physically hurt by a robot or a doctor who uses AI, or discriminated against by an automated hiring or lending system, the injured parties may end up in a courtroom. In these cases, the court is tasked with determining who is responsible for the injury, if anyone, and accordingly whether a defendant should incur liability. The technology itself plays a central role in the analysis. But if the court treats the technology as if the configuration presented was the only one possible, and examining it only as a black box, it fails in its role.

What we know as the black box is the result of a formal process in computer science known as abstraction. Abstraction is the process of separating complex systems into isolated components that can be completely described by their inputs, outputs, and the relationship between them, without needing to know anything about the internal operation. It is a technique to manage information, without which it would be almost impossible to build systems of any meaningful complexity. In a less formal sense, we all use abstractions in our daily lives—any concept we hold can be considered an abstraction—but the formal process of abstraction is a

[1] *E.g.* FRANK PASQUALE, THE BLACK BOX SOCIETY (2015); Danielle Keats Citron & Frank Pasquale, *The Scored Society: Due Process for Automated Predictions*, 89 Wash. L. Rev. 1, 33 (2014); Sonia K. Katyal, *The Paradox of Source Code Secrecy*, 104 Cornell L. Rev. 1183, 1186 (2019); Frank Pasquale, Restoring Transparency to Automated Authority, 9 J. ON TELECOMM. & HIGH TECH. L. 235, 236-37 (2011) (recounting the origins of using trade-secret protections for algorithms); Brenda Reddix-Smalls, *Credit Scoring and Trade Secrecy: An Algorithmic Quagmire or How the Lack of Transparency in Complex Financial Models Scuttled the Finance Market*, 12 U.C. DAVIS BUS. L.J. 87, 88-90 (2011).

[2]; *E.g.* Ashley Deeks, *The Judicial Demand for Explainable Artificial Intelligence*, 119 Colum. L. Rev. 1829 (2019); Sandra Wachter et. al., *Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the Gdpr*, 31 Harv. J.L. & Tech. 841, 843 (2018); Cary Coglianese & David Lehr, *Transparency and Algorithmic Governance*, 71 Admin. L. Rev. 1, 4 (2019).

[3] *E.g.* W. Nicholson Price II, *Black-Box Medicine*, 28 Harv. J.L. & Tech. 419, 434 (2015); Charlotte A. Tschider, *Beyond the "Black Box"*, 98 Denv. L. Rev. 683 (2021); Emily Berman, *A Government of Laws and Not of Machines*, 98 B.U. L. Rev. 1277, 1315 (2018); [[others]]

[4] A notable exception is David Lehr and Paul Ohm's breakdown of the creation of the machine learning process that functions as a primer for legal scholars. David Lehr & Paul Ohm, *Playing with the Data: What Legal Scholars Should Learn About Machine Learning*, 51 U.C. Davis L. Rev. 653, 656 (2017).

foundational methodological component of computer science, often taught on the first day of the introductory class.[5]

Abstraction serves to separate a subsystem from the rest of the system, and a device from the rest of the world. As such, the choices that go into an abstraction—the types and definition of inputs and outputs, the specification of the operation that converts inputs to outputs—affect, and often dictate, how the world sees the object in question and how relationship to the object are formed. Speaking about technology generally, sociologist Madeleine Akrich famously referred to "scripts" that innovators imbue technologies with, inscribing their "vision of (or prediction about) the world in the technical content of the new object."[6] She observed that:

> [C]hoices made by designers take the form of decisions about what should be delegated to whom or what, [meaning] that technical objects contain and produce a specific geography of responsibilities, or more generally, of causes. To be sure this geography is open to question and may be resisted. Nevertheless, it suggests that new technologies may not only lead to new arrangements of people and things. They may, in addition, generate and "naturalize" new forms and orders of causality and, indeed, new forms of knowledge about the world. . . . [T]echnologies may generate both forms of knowledge and moral judgments.

Abstraction creates such a script, imbuing the technology with a viewpoint about what the boundary of the technology is and who should be responsible for which outcomes. The black box is considered a black box precisely because the purpose of it is to hide the details of what's going on inside and to insulate those concerned with the internal parts from any external context. When the developer puts an algorithmic system in a black box, she is defining the object, implicitly telling the user that they need not worry about its contents, and telling the world that anything outside the box is not her problem.[7] This communicative aspect is an unavoidable byproduct of creating the technology. Abstraction is necessary, abstraction choices are necessary, and thus such scripts are inevitable.

Courts encounter technologies after injuries occur, long after the abstraction choices are made and rendered invisible by hindsight. At that

[5] *See, e.g.* MIT Open Courseware, 6.001: Structure and Interpretation of Computer Programs Class 1 Lecture Notes, https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-001-structure-and-interpretation-of-computer-programs-spring-2005/lecture-notes/lecture1webhand.pdf

[6] Madeleine Akrich, *The De-Scription of Technical Objects*, *in* SHAPING TECHNOLOGY/BUILDING SOCIETY 205, 208 (Wiebe Bijke & John Law, eds. 1992).

[7] *See* Kirsten Martin, *Ethical Implications and Accountability of Algorithms*, at 8-9, J. BUS ETHICS (2018) (describing how algorithm design "prescribes the delegation of responsibilities in decisions").

point, stating the facts of what happened and who is responsible for what becomes the province of the court. Courts today know that they are hearing from different parties: plaintiffs, defendants, and witnesses. But they must also realize that they are hearing from the technologies themselves. The choices embedded within the technologies make claims as to the facts and as to the lines of responsibility, claims that compete with those very same things that the courts must decide. But these choices were made by engineers and product managers, for the purpose of efficiency, portability, and profit. These are not the primary concerns of a courtroom; rather, courts are in the business of ensuring that lines of responsibility match those normative concerns that society has decided to enshrine in law. If courts uncritically accept the claims embedded within the technologies, they are abdicating their roles. Their rulings will adopt technologists' decisions, giving them an often-unwarranted legal imprimatur. Legal actors are the only ones who can who construct black boxes in the courtroom.

This Article proceeds in three parts. Part I introduces the formal concept of abstraction in computer science: how it is used, how abstraction boundaries are chosen, and how they are subsequently rendered invisible. Part II turns to law, examining how abstraction choices can change outcomes in tort law, discrimination, and factfinding more generally. Finally, Part III offers another way, demonstrates with three hypothetical case studies that once judges resurface and critique the abstraction boundaries, the problems of machine learning black boxes reduce to familiar problems that courts know how to resolve.

I. DECONSTRUCTING THE BLACK BOX

Abstraction in computer science is a formal process that creates boundaries between the inside and outside of a technical component of a system. Abstraction also has the effect of defining the system as an object that exists separately from the rest of the world, and delineating lines of responsibility: the designer is responsible for what happens inside the abstraction boundary, and the user should never have to know how the inside works. In this Part, we describe formal abstraction processes, and the ways in which abstraction choices can be read as factual claims about the boundaries of a system and normative claims about where responsibility lies.

A. Formal Abstraction

Abstraction is perhaps the most foundational idea in computer science, and more broadly in the design of digital technology. It is the intellectual foundation that allows a designer to understand a system in terms of its smaller components, building up to incredible complexity, by restricting her view to a particular part of the system any time. Formal abstraction is the technique of specifying a system solely in terms of a set of well-defined inputs, a set of well-defined outputs, and the relationship that transforms inputs to outputs. The beauty of formal abstraction is that one need not know—and indeed should not care—how the transformation from inputs to outputs occurs, as long as it works.

For the simplest of examples,[8] imagine a program `square` that computes the square of a number. Here, the input is a number *x* and the output is a number *y* which corresponds to the input *x* squared. The module could be represented in code in a number of ways. Here are two:

```
1) define square(x) {
       y = x * x;
       output y
   }

2) define square(x) {
       y = 0;
       for i = 1 to x:
          y = y + x;
       output y
   }
```

These snippets of code will both compute $y = x^2$ for any integer x. The point of abstraction is that someone can use the `square` module or subroutine, know that it will reliably return the value they seek, and never have to know what the code inside looks like, as long as it reliably computes the function it is supposed to on the inputs it is designed for. That is powerful. Nonetheless, the developer of `square` might care very much about how it is implemented; one version might be more efficient in terms of compute power or time, or one might be harder to debug.

These abstractions are then layered into what we call systems. We can imagine a second abstraction called `cube` that computes the cube of a number. If you wanted to create a system that took in two inputs *x* and *y* and returned $z = x^2 + y^3$, you could create a new abstraction `foo` that takes two inputs, *x* and *y*, one output *z*, and a transformation function built using the `square` and `cube` blocks:

```
define foo(x,y) {
      z = square(x) + cube(y);
      output z
```

[8] This example is taken from HAROLD ABELSON, GERALD JAY SUSSMAN & JULIE SUSSMAN, STRUCTURE & INTERPRETATION OF COMPUTER PROGRAMS (1985).

```
    }
```

A major benefit of abstraction is that the creator of `foo` does not need to think about how `square` and `cube` are made. The details are "abstracted away." Equally important is that the creators of `square` and `cube` do not need to think about how they will be used. The context, too, is abstracted away. This is how abstractions become subcomponents in bigger systems. It's abstractions all the way down.

When a developer encounters a problem to solve, their first step is therefore to identify the necessary inputs, outputs, and the transformation with the desired properties. This becomes, in computer science parlance, the "problem definition." This defines what the developer is responsible for in that moment. Subsequent activities focus on methods (or "algorithms") to transform the inputs to the desired output as effectively as possible. If at any point in the development process the specification of inputs and outputs change, that becomes a new problem, and the cycle starts again. The decisions about what constitutes an input and what constitutes a desirable output are the "abstraction choices" made as part of the design of the resulting system.

These choices also represent a boundary between the interior of a system and its exterior. For the designer of the system, the processes that lead to the production of the inputs as well as the processes that follow after the production of the outputs are not considered part of the problem space and are disregarded. (The creator of `square` need not know that someone at a later time will come along and use it to make `foo`.) Correspondingly, for a user of the system, the internal design choices used to construct the system are abstracted away: the user does not need to know how the input is transformed to the output as long as it does so correctly. (The creator of `foo` need not know the choices that went into making `square`.)

The math examples above can illustrate the concept of abstraction, but they are a little too simple to demonstrate the stakes of abstraction choices. For a more tangible example, consider the abstraction of a coffee maker.[9] A reasonable abstraction of the problem (making coffee) is to assume as input water and coffee beans, and as output a cup of coffee. This problem definition does not concern itself with where the water came from, whether the beans were sourced from an organic grower, or even what size the user's mug is; any issues concerning the input or output conditions are not the designer's problem. Conversely, a user of this coffee machine operates only on the outside of this system. For the user to brew a delicious cup of coffee, "you

[9] This example, too has been used elsewhere. *See* James W. Malazita & Korryn Resetar, *Infrastructures of abstraction: how computer science education produces anti-political subjects*, 30 *Digital Creativity* 300 (2019); Thorben Janssen, OOP Concept for Beginners: What is Abstraction? https://stackify.com/oop-concept-abstraction/

need to provide water and coffee beans, switch it on and select the kind of coffee you want to get."[10] The user need not know anything about how the machine works: someone else worried about that and created a coffee machine that now acts as an abstraction and hides all these details.

There are several consequences to this abstraction. As stated above, it allows the designers to think about the complex system in a way that allows it to be built more reliably, or even built at all. In this Article, we focus on two specific conceptual consequences of creating this abstraction. One is an epistemic result. We now have a concept labeled "coffee maker" that we did not have before the abstraction was created. We can now refer to the coffee maker as a thing and we can understand it as one. The conceptual space that the "thing" takes up is delineated by its abstraction boundary. The internal parts of the coffee machine, such as the heating element or the mechanism to move water from the tank to the coffee grinds, are all internal to the abstraction. The rest of the world is external. The machine is a conceptual boundary. The second effect is a division of responsibility. What is inside the machine is different from what is outside. If coffee comes out badly because the heating element doesn't work well, the manufacturer is to blame; if a user inputs foul-tasting water, then it's not the manufacturer's problem.

Choosing the abstraction of a problem is a choice which has consequences, which can even be seen in this simple example. The abstraction boundary for the coffee machine—between the user providing the supplies and the machine that uses them—may seem natural or obvious, but it's not. For example, there is a difference between a coffee machine that takes in pre-ground coffee and one built with a grinder that can takes in whole beans. Conceptually, whether the grinding is a function internal or external to the coffee machine changes, whether the grinder is a component of the machine changes, and the responsibility for the quality of the grind changes with it as well. Another example is how strictly the inputs are defined. If a coffee machine abstraction only accepts filtered water as an input, then if tap water is used as the coffee tastes bad as a result, that falls on the user; but if the input is merely defined as "water," then the coffee maker manufacturer can reasonably be blamed for the bad tap water coffee. Finally, the output must be specified as well. Some coffee machines dispense coffee into a mug provided by the user and some provide their own pot that the user can later dispense from. If the coffee maker dispenses into a mug, the space provided will dictate what types of mugs can be used, so the mug shape becomes another input to the abstraction. Basically, any abstraction can be created and many versions can be defended for different reasons, but the abstraction

[10] Thorben Janssen, OOP Concept for Beginners: What is Abstraction? https://stackify.com/oop-concept-abstraction/

choices change what counts as the object and who bears responsibility for which parts.

This brings us to machine learning. Portable forms of problem definitions and abstraction choices make up the foundations of machine learning technology. For instance, the "supervised learning task", or the problem of building a predictive model, can be solved in many different ways, but is defined by a particular abstraction:[11] The input to the task is a training data set, or a list of instances *X* of some data domain, along with exactly one output *y* for each instance. The output of the learning task is a function relating *X*'s to *y*'s. The function should accurately calculate predictions on unseen data. For example, the data domain could be "emails," and each email could be assigned a label of "spam" or "not spam."; this setting, in which outputs must be discrete, is an instance of the supervised learning task called *classification*. Of course, there is much left unspecified by this abstraction about how to solve the supervised learning task. What types of functions should be considered? How should a function's performance be estimated? What assumptions must be made about the probabilistic distribution of the incoming data? These are all parts of the abstraction—in real examples, fully specifying an abstraction can be complicated, but abstractions nonetheless provide utility.

Popular solutions to the supervised learning task are implemented by the scikit-learn Python package[12]. Each solution is implemented by different functions within the package, but can be presented to a programmer looking to build their own machine learning model through a unified framework. For instance, though a logistic regression solution and a decision tree solution are trained based on differing assumptions about the data, each has a fit method which only takes in training data. Here, we can see how simple it is to train the two models with very similar lines of code.

```
1) tree_model = tree.DecisionTreeClassifier()
   tree_model.fit(X,y)

2) lr_model = linear_model.LogisticRegression()
   lr_model.fit(X,y)
```

Since this abstraction is so portable, most solutions to the classification problem can be "plugged in" to many real-life problems without much extra work, such as determining whether or not an x-ray image of a lung contains a cancerous tumor, or which character in the English alphabet an image of a handwritten character displays. Thus, scikit-learn saves the programmer the work of actually implementing these classification solutions, and in turn

[11] GARETH M. JAMES, ET AL., AN INTRODUCTION TO STATISTICAL LEARNING 6 (2013)
[12] http://scikit-learn.org/

guarantees that the results will be correct according to their specifications. The developer is thus alleviated of the responsibility of performing that task. Importantly, the package functions used to train the model remain conceptually separated from the model itself.

While scikit-learn takes much of the responsibility of building a good machine learning model out of the hands of the programmer, the documentation and behavior of the technology makes it clear that there are elements of the resulting technology that are outside of the responsibility of the package. Since the training data is an input, the package assumes that data is correct; the classification learning task as understood by mathematicians does not account for issues of measurement or corruption. It is the programmer's job, not the package's, to validate this. It is also the programmer's job to select which implementation of the classification problem to use; since they may perform differently for different problem settings, it is up to the programmer to pick one that is appropriate. For more advanced users, many knobs are available for a programmer to adjust how the learning procedure is implemented; however, the developers of scikit-learn provided default settings, implicitly accepting some responsibility for the model in that scenario. In this case, this responsibility for reasonable default behavior was also noted explicitly by the developers of scikit-learn. Another way of understanding this is that the scikit-learn package abstracts away the learning task, but is only a component of the larger system abstraction, with its own definitions chosen by the programmers.

B. Abstraction Boundaries and Choice

Choices of abstraction boundary are motivated by a number of considerations. They may be based on physical constraints, aesthetic considerations, or economic concerns. Usually, in computer science, choices about abstraction boundaries are motivated by the ideals of portability and efficiency[13]. One solution that can be "plugged into" a group of similar problems is preferred to the parallel development of multiple individual solutions which share components or similarities. We can even apply these design principles in the coffee machine example. A machine designer might decide that a coffee machine with its own grinder is more "portable" because of easy access to beans, or that a machine that takes in pre-ground beans is

[13] *See* Andrew D. Selbst, et al., *Fairness and Abstraction in Sociotechnical Systems,* PROC. 2019 ACM CONFERENCE ON FAIRNESS, ABSTRACTION AND TRANSPARENCY 59.

more efficient because the designer no longer has the problem of syncing the grinding and brewing processes.[14]

While portability and efficiency are fundamental imperatives motivating computer scientists when designing systems, other factors can be—and regularly are—taken into account. Indeed, designers can (and often do) also choose to define their abstraction boundary in such a way as to account for the context in which the tools they develop will be used. The broader computer science community has been actively researching the complexities of human-computer interaction, or HCI, for decades. By the 1980s, the software industry was no longer simply composed of technically trained programmers; individuals were being hired as "usability professionals" to make sure the exteriors of the tools they developed were intuitive and effective for users.[15] Methods to evaluate the usability of a tool range from informal or heuristic inspections to formal metrics to empirical methods such as user testing.[16] For this reason, when new technology is introduced, researchers often try to anticipate interactions that could cause adverse outcomes.

To put this in terms of abstraction definition, recall that an abstraction requires that inputs be "well-defined." For example, a program may require data to be structured in a certain way in order to get a result. Anything structured differently is not well-defined in the abstraction and the system will either do nothing or give an incorrect output. But when the program requests input from a user, the user would not know anything about the required structure of data internally, and the system will need to ensure data is entered correctly. A trivial example would be entering a date into a webpage. None of us looks at the code when using a webpage; instead, the user interface will either require the user to enter a date in a given format or have a drop-down menu to ensure the input to the internal abstraction is correct. Whether the input from the user comes from a drop-down menu or an open field to type numbers in is part of the "website" abstraction, but will often be chosen for reasons more to do with the expected users. If a website expects entries from just the United States, it may just expect a "mm/dd/yyyy" field entry, but it might be confusing for a global website where others use a "dd/mm/yyyy"

[14] Note here that this determination of efficiency comes from the perspective of the machine maker rather than the world as a whole. Someone still has to grind the beans, but putting the grinding outside the abstraction makes it more efficient for *the maker* because it is someone else's problem. Some efficiency gains may reduce cost overall, while some reduce cost only for the creator of the machine. We will return to this theme.

[15] Deborah J. Mayhew, *User Experience Design: The Evolution of a Multi-Disciplinary Approach*, 3 J. OF USABILITY STUDIES 99 (2008).

[16] Jakob Nielsen, *Usability Inspection Methods*, PROC. CHI '94: CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS 413 (1994).

scheme, so the choice of how to define the inputs at the user interface level depends on what is expected about the user.[17]

Many successful systems built upon machine learning algorithms are, in fact, designed explicitly to account for human interaction; concerns about the user experience are incorporated directly into the interior of the abstraction boundary. Some good examples can be drawn from the field of natural language processing (NLP). Alexa Conversations is a framework "for building goal-oriented dialogue systems that is scalable, extensible, as well as data efficient"[18] developed at Amazon. The paper describing the tool describes how end-users using the eventual dialogue system may interact with the technology; they specifically design to accommodate "natural conversational phenomena like entity sharing across turns or users changing their mind during conversation."[19] They give an example of ordering pizza, where user correction ("actually, make it a small") is anticipated. Researchers at Spotify have also conducted research on how to identify user motivations for making nonspecific queries in an interactive dialogue system, in an attempt to better anticipate their actions and needs.[20]Systems are not always designed to work in a vacuum; they can be designed to work well in context.

Designers can also take into account the fact that users can learn to use and reason about tools over time. A good example is SepsisWatch, a tool developed to rank the probability of sepsis, a leading cause of death in hospitals, among patients in a dashboard for physicians. Mark Sendak and colleagues studied the integration of the tool into the Duke University Hospital, finding that:

> [N]urses developed expertise and practices over time that contextualized the information displayed in Sepsis Watch, and facilitated the integration of the tool into existing clinical practice. . . . For instance, RRT nurses developed a practice of working outside of the app and opening a patient's EHR chart before calling

[17] Worrying about inputs is especially common from the perspective of security: designers and researchers must use threat modeling to attempt to anticipate adversarial interactions that could compromise the privacy of systems. A common type of security intrusion is known as a SQL injection. This is where an attacker creates an input to a database that is not in the expected form, and the database spits out information that it is not supposed to. How to properly handle inputs that are not well-defined is therefore an important part of security. And more broadly, methods to make threat modeling more effective is also an active area of research. *See e.g.* Jane Cleland-Huang, et al., *Keeping Ahead of Our Adversaries,* 33 IEEE Software, 24 (2016).

[18] A. Acharya, *Alexa Conversations: An Extensible Data-driven Approach for Building Task-oriented Dialogue Systems*, in PROCEEDINGS OF NAACL-HLT (2021).

[19] *Id.* at 1.

[20] Jennifer Thom, Angela Nazarian, Ruth Brillman, Henriette Cramer, & Sarah Mennicken, *"Play Music": User Motivations and Expectations for Non-specific Voice Queries*, in PROC. ISMIR (2020)

> the ED physician. By reading through a patient's chart, the RRT nurse was preparing to present the full clinical picture and do 'due diligence,' in the words of one interviewee, in anticipation of questions received from physicians.[21]

As the developers of SepsisWatch proceed with maintaining the tool, they can now incorporate this additional information about how it is used, potentially resulting in new specifications for the system.

These examples illustrate that, even in the context of machine learning systems, abstraction boundaries can be drawn in a number of different ways. Developers can incorporate different amounts of context surrounding a technical task into the design of an algorithm, resulting in an abstraction boundary that takes responsibility for a large and complex problem specification; or, they can choose to only address a simpler, purely technical task with a more limited problem specification. Importantly, just because a designer chose a certain abstraction boundary does not mean that it was the correct or only one that could have been used to inform the design of the technology.

C. The Invisibility of Abstraction Choices

Abstraction choices, once made, often fade into the background and become invisible, rendering an explicit discussion about the design choices difficult. While abstraction choices are fundamental, they are not typically documented or made explicit in a formal way. Indeed, it is not obvious from inspection of an already-designed system where the abstraction choices were made and how. Yet the choices have a profound impact, by virtue of placing the boundaries, on how the system might be expected to behave under "normal" versus "abnormal" conditions, and what "normal" even means.

Consider the current concerns around the use of facial recognition and analysis in various settings like video interviews, remote proctoring, and surveillance. In all these settings, the standard abstraction choices are expressed in terms of a general *technical* task: whether a face is being matched to a given target image, a database, or even whether a face is expressing some kind of emotional affect from which intent can be inferred. Importantly, the

[21] Mark Sendak, et al., *The Human Body Is a Black Box: Supporting Clinical Decision-Making with Deep Learning*, PROC. 2020 CONF. ON FAIRNESS, ACCOUNTABILITY, AND TRANSPARENCY, 99, 106.

source of the facial scans and the purpose of the match is outside the abstraction boundary and is therefore ignored. The concept is the technology.

This is, however, not a *fait accompli*, but an explicit design choice. Consider three different scenarios where one might choose to use a system that matches faces to a database. One system might be in use at an amusement park to allow visitors access to rides. Another system might be in place at a bank to authenticate account holders prior to transactions. A third system might be in used by the TSA to match returning travelers to their entries in an immigration database. Imagine now that the abstraction boundary had been designed to incorporate these specific applications. We would end up with three distinct problems—call them amusement park access, bank authentication, and TSA-verification. It is quite likely that the operating conditions for the amusement park access system might involve testing for occluded faces, faces wearing painted-on decorations or silly hats, or even sunglasses. Bank authentication systems might expect an individual to place their face in a very structured setting and have a high bar for a match, because there is a fallback option of entering an ATM PIN or some other password. TSA verification systems might be implemented inside a specific device that can be manipulated so that the image of an individual appears exactly within a prescribed frame, and can be cross checked against traveler information provided via a boarding pass scan. The systems may have similar underlying technology or not. Today, because we use the concept "facial recognition" constantly, we tend to think of them as applications of the same underlying technology, but if the abstraction boundary were drawn wider to encompass the contexts, we would not have any reason to associate them with each other. The difference in thinking of each of these systems as entirely different is that the underlying class of systems known as "facial recognition" never enters our consciousness. Because the abstraction boundary was drawn as it was, we use that concept instead. Either way, the choice of abstraction boundary fades into the background.[22]

II. CONSTRUCTING THE BLACK BOX IN COURT

As noted above, abstractions have important ramifications. The creation of any technology creates what Akrich called "scripts" that "generate both forms of knowledge and moral judgments."[23] That is, the creation of the abstraction boundary can be seen as making factual and normative claims

[22] *Cf.* GEOFFREY C. BOWKER & SUSAN LEIGH STAR, SORTING THINGS OUT __ (1999) (noting that classifications in general are necessary then recede into the background)

[23] Madeleine Akrich, *The De-Scription of Technical Objects*, *in* SHAPING TECHNOLOGY/BUILDING SOCIETY 205, 207 (Wiebe E Bijker & John Law, eds 1997).

about what the object is and where lines of responsibility fall. From the engineers' perspectives, they are not making such claims at all. The choice of abstraction boundary is usually a design choice governed by concerns such as efficiency and portability, economic and marketing concerns, and legal requirements, where they exist. The engineers are not seeking to make either epistemic or normative claims, and it is important to emphasize that these scripts are implied by the technology itself, not by statements of the actors involved.[24] Nonetheless, the abstraction choices do encode the considerations that were taken into account.

Once an injury occurs that lands the user of creator of a machine learning algorithm in court, it becomes the role of legal actors to do the relevant factfinding and demarcate lines of responsibility. But at that point, the technology has been established and is in use, the factual and normative claims made by the creation of the technology are set, and they have receded into the background. While these invisible factual and legal claims are chronologically prior to legal consideration, they are and must be secondary to legal actors' determinations in importance. Thus, when encountering these technologies, it is important that courts resurface the factual and normative claims being made by the technologies themselves and interrogate them, or risk allowing technologists to (unintentionally) usurp their function.

In this Part, we discuss the implications of these hidden abstraction choices for tort law, with a focus on foreseeability, for anti-discrimination law, and for factual determinations more generally. Ultimately, the legal actors are the factfinders and the people who apportion responsibility, and it is therefore they, in the legal setting, who get to decide what a "correct" abstraction boundary is from the standpoint of the law.

A. Abstraction Choices and Tort Law

At a high level of generality, tort cases have the same overarching superstructure: A plaintiff was injured, and the case is about figuring out whether the injury can be properly attributed to a defendant or some subset of multiple defendants. In this sense, almost all of tort law can be seen as a question of how to assign responsibility.

Most of tort law—specifically negligence and products liability—centers on determinations of reasonableness. In the case of an injury resulting from the use of a machine learning system, if the defendant is the user of a system, the query will depend on whether they behaved reasonably in using

[24] Martin, *supra* note 7, at 6 ("Scripts are durable, and the technology's script becomes independent of the innovator once in use".)

the device,[25] and if the defendant is the device's creator, whether the creation of the device was unreasonably dangerous.[26] But tort carries with it an important limitation to liability: foreseeability. It is a general rule of liability that "a defendant is responsible for and only for such harm as he could reasonably have foreseen and prevented."[27] Based on this notion, the concept of "reasonable foreseeability"—for which "foreseeability" is usually just a shorthand—pervades liability regimes.[28] In tort, it is most famously associated with negligence, but it plays a role in product liability and even strict liability as well.

The "black box" metaphor in legal discourse surrounding machine learning and AI is usually used to imply an element of unknowability, and corresponding unforeseeability.[29] For example, Ryan Calo has argued that emergent behavior of AI and especially the interactions between machines,

---

[25] Restatement (Third) of Torts: Phys. & Emot. Harm § 3 (2010); *see also* Andrew D. Selbst, *Negligence and AI's Human Users*, 100 B.U. L. REV. 1315 (2020)

[26] Restatement (Second) of Torts § 402A; *see also* Selbst, *supra* note 25 at __.

[27] H.L.A. HART & TONY HONORÉ, CAUSATION IN THE LAW 255 (2d ed. 1985).

[28] Shyamkrishna Balganesh, *Foreseeability and Copyright Incentives*, 122 HARV. L. REV. 1569, 1591 (2009).

[29] *See* Ryan Calo, *Robotics and the Lessons of Cyberlaw*, 103 Cal. L. Rev. 513, 515 (2015); *id.* at 542 (("[T]he mechanisms by which the law sorts fault involve deeply human concepts such as . . . foreseeability . . . which are absent where a system is built to be unpredictable by design."); Jason Millar & Ian Kerr, *Delegation, Relinquishment, and Responsibility: The Prospect of Expert Robots*, *in* ROBOT LAW 102, 107 (RYAN CALO, A. MICHAEL FROOMKIN & IAN KERR EDS., 2016) (calling AI "unpredictable by design"); Matthew U. Scherer, *Regulating Artificial Intelligence Systems: Risks, Challenges, Competencies, And Strategies*, 29 Harv. J.L. & Tech. 353, 365 (2016) ("AI systems can develop solutions that "may not have been foreseeable to a human– even the human that designed the AI."); Weston Kowert, Note, *The Foreseeability of Human–Artificial Intelligence Interactions*, 96 TEX. L. REV. 181, 183 (2017) (When [a] software developer places [an] artificial intelligence into the real world, the developer can-not predict how the artificial intelligence will solve the tasks and problems it encounters."); Kenneth S. Abraham & Robert L. Rabin, *Automated Vehicles and Manufacturer Responsibility for Accidents: A New Legal Regime for a New Era*, 105 VA. L. REV. 127, 134 (2019); FRANK PASQUALE, THE BLACK BOX SOCIETY (2015); Yavar Bathaee, *The Artificial Intelligence Black Box and The Failure of Intent and Causation*, 31 HARV. J. L. TECH. 889, 907 (2018) ("Black-box AI may do things in ways the creators of the AI may not understand or be able to predict."); W. Nicholson Price II, *Regulating Black-Box Medicine,* 116 MICH. L. REV. 421 (2017) at 433 (discussing the opacity of medical black box algorithms); Chris Reed, How Should we Regulate Artificial Intelligence?, ROYAL SOC'Y PUBL'G, Apr. 29, 2018, at 1; https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2017.0360; David Nersessian, JD, PhD & Ruben Mancha, PhD, *From Automation to Autonomy: Legal and Ethical Responsibility Gaps in Artificial Intelligence Innovation*, 27 Mich. Tech. L. Rev. 55, 70 (2020) ("[T]he liability risks associated with certain types of AI will be inherently unforeseeable."); Omri Rachum-Twaig, *Whose Robot Is It Anyway?: Liability for Artificial-Intelligence-Based Robots*, 2020 U. Ill. L. Rev. 1141, 1152–53 (2020)

will sometimes "legitimately surprise all involved,"[30] Judge Curtis Karnow has argued that the complexity of machine learning systems may cause them to evade all tort liability,[31] and Mark Lemley and Bryan Casey have argued that the unpredictability of machine learning technologies can lead to unforeseen harms.[32] While certainly not everyone argues that all machine learning harms are unforeseeable,[33] it is not always clear how to approach the foreseeability question.

Claims as to the unforeseeability of AI tend to treat foreseeability as primarily a factual question, whether a certain event was capable of being foreseen. But foreseeability is not an objective factual claim. Functionally, it is a doctrinal move courts make to decide where to cut off liability. The purpose of this limitation depends on purpose one assigns to liability rules in general. If one sees liability allocating responsibility for past actions, delimiting the boundaries of a legal "wrong,"[34] foreseeability marks the outer bound of culpability.[35] If a certain outcome cannot be foreseen, it would usually be unfair to hold a person legally responsible for failing to account for that possibility when choosing their course of conduct. If, however, one sees the purpose of liability rules as creating incentives to shape future conduct, then foreseeability acts as a limitation to the incentives that such deterrence can create. If a certain consequence is not a foreseeable result of a course of action, then attaching a penalty to that consequence can have no effect on the incentives to engage in that course of action.[36] For either practical purpose, the doctrinal approach to foreseeability is the same: a judgment that some outcome is unforeseeable sets that eventuality outside the scope of a defendant's responsibility.

Foreseeability can also be seen as a practical response to the fact of "bounded rationality."[37] Bounded rationality as a concept holds that when making decisions in complex and uncertain circumstances, people can only

[30] Calo, *supra* note 29, at 555.

[31] Curtis E.A. Karnow, *The Application of Traditional Tort Theory to Embodied Machine Intelligence*, *in* ROBOT LAW 51, 74 (Ryan Calo, A. Michael Froomkin & Ian Kerr eds., 2016)

[32] Mark A. Lemley & Bryan Casey, *Remedies for Robots*, 86 U. Chi. L. Rev. 1311, 1334 (2019).

[33] Mark A. Geistfeld, *A Roadmap for Autonomous Vehicles: State Tort Liability, Automobile Insurance, and Federal Safety Regulation*, 105 CALIF. L. REV. 1611 (2017), F. Patrick Hubbard, *"Sophisticated Robots": Balancing Liability, Regulation, and Innovation*, 66 FLA. L. REV. 1803, 1854 (2014); [[others]]

[34] *See, e.g.*, Benjamin C. Zipursky & John C.P. Goldberg, *Torts as Wrongs*, 88 TEX. L. REV. 917 (2010).

[35] *See* David G. Owen, *Figuring Foreseeability*, 44 WAKE FOREST L. REV. 1277, 1277–78 (2009).

[36] *See* Balganesh, *supra* note 28, at __

[37] *Id.* at 1592.

take a finite number of factors and factual premises into account.[38] While every action has an unlimited string of effects that it changes in the world, a person can only contemplate so many consequences in reality. Thus, foreseeability combines a descriptive observation and normative concern: A person will only be able to process so many potential outcomes from their actions, and as a result, foreseeability gives the law a way to hold a person only responsible for that subset that it is fair to expect someone to contemplate. Whether it is fair for someone to contemplate a given outcome is not just a function of its likelihood, however. A potentially devastating consequence of someone's actions may be held foreseeable even where it less likely than other mundane consequences.

Foreseeability is frustrating as a concept because it is billed as a legal test, but it is truly a policy judgment top to bottom. As a test, it appears endlessly malleable, impossible to pin down.[39] Case law on foreseeability is all over the place, and courts are wildly inconsistent in how the doctrine is to be applied.[40] The paradox of foreseeability is that any dreamed-of consequence that can be stated can in some sense be foreseen, but that is not what is meant by the idea.[41] But frustrating as it may be, it is precisely this malleability that gives foreseeability its importance in shaping the limits on legal liability. Judges and juries making foreseeability determinations are not making factual claims about what literally can or cannot be seen by a hypothetical neutral third party. Instead, they are making policy judgments about whether the situation presented by the facts in the case should be outside the bounds of what we will hold people responsible for. Foreseeability fails at being a legal test because it isn't one; it is a judgment about where liability should end.[42]

Reasonable foreseeability canonically contains two ideas: an assessment of "objective" probability of a given event occurring and a claim as to what the reasonable person would know about that probability.[43] Foreseeability is therefore the joining of an unstated factual claim of probability and a judgment as to what the reasonable person would know about such a probability. Sometimes referred to as an "epistemic

[38] See Amos Tversky & Daniel Kahneman, *Judgment Under Uncertainty: Heuristics and Biases*, 185 SCI. 1124 (1974).

[39] Owen, *supra* note 35, at 1279 ("As a test of responsibility for consequences … foreseeability may seem almost empty of content, so devoid of substantive meaning as to mock the concept of a rule, principle, or standard for evaluating conduct or its consequences to determine responsibility in tort.")

[40] Benjamin C. Zipursky, *Foreseeability in Breach, Duty, and Proximate Cause*, 44 Wake Forest L. Rev. 1247, 1260 (2009) (describing courts' inconsistency in no-duty foreseeability rulings)

[41] HART & HONORE, *supra* note 27, at 256–27 ("[I]n one sense everything is foreseeable, in another sense nothing.")

[42] *See* Owen, *supra* 35, at 1279.

[43] W. Jonathan Cardi, *Reconstructing Foreseeability*, 46 B.C. L. Rev. 921, 938 (2005)

probability,"[44] such a judgment is inherently normative; it is a decision about what the reasonable person should know. Thus, it is no surprise that foreseeability is not simply a factual claim, but a matter of judgement.

While this standard description of foreseeability would suggest that there is an objective portion and a subjective portion, the description is incomplete. Rather, both halves of the foreseeability inquiry rely on subjective judgment. As Jonathan Cardi explains:

> The objective probability that an event will occur is best understood as the event's relative frequency within a reference class of events—for example, the relative frequency of a car crashing when one drives thirty miles per hour over the speed limit. . . .
>
> The determination of foreseeability requires two levels of judgment. First, the relevant decisionmaker must properly describe the subject event and frame that event within its proper reference class of events. Continuing the example from the prior paragraph, an event might be described as "injury," "injury from a car crash," "shattered pelvis from a car crash," or "shattered pelvis from a car crashing into a tree." The spectrum of possible descriptions, from general to specific, is quite broad. Similarly, the reference class of events might be described as "while driving," "while speeding," "while driving thirty miles over the speed limit," "while driving thirty miles over the speed limit on a rainy day," or any number of other possible variations. One's choice of description of the event and of the reference class of events strongly influences the event's relative frequency.[45]

Thus, it is only after the framing of the question that the factual probability is even conceptually defined—though the probability itself is still unknown and likely unknowable in reality.

This framing of the problem is an important first step in the analysis, for which law provides no guidance.[46] The framing circumscribes the analysis for the ultimate decisionmaker, whether judge or jury. We all likely agree that a reasonable person would foresee an "injury…while driving thirty miles over the speed limit on a rainy day," while such a person would be less likely to foresee a "shattered pelvis from a car crashing into a tree…while driving." In extreme cases, the framing essentially determines the result in full.[47] With a less extreme framing, the decision is still anchored, but the judgment of foreseeability becomes a closer call, truly made by the formal decisionmaker—usually a jury.

[44] *Id.*

[45] Cardi, *supra* note 43. at 939.

[46] *Id.*

[47] Juries can be unpredictable, but the ability for a judge to take facts away from the jury in extreme cases means that it is possible for an extreme frame to determine the outcome.

The second set of judgments is the familiar application of foreseeability to the facts. Here, the jury or judge just makes an informed judgment call. The common law approach to questions for which there are not good analytical answers—especially fact-laden judgments—is to delegate them to a decisionmaker or factfinder. At that point the interesting analytical question is not in what the outcome is—because if we could answer that we wouldn't be sending it to the judge or jury—but rather what types of arguments or evidence can be taken into account in the foreseeability determination. In different legal contexts, the law asks whether a different aspect of the facts was foreseeable. In negligence law, foreseeability in the breach element asks whether a particular risk is a foreseeable consequence of a certain behavior, such that the possibility of the risk coming to fruition should be considered by a person when choosing the level of care to exercise, while in product liability the question may be whether a particular use of the product was foreseeable. Thus, the specific type of foreseeability inquiry will account for different types of factual evidence. More than that, though, foreseeability inquiries exist within a given social and legal context. The facts that pertain to the question are "external to the idea of foreseeability itself," and instead depend on "the different policies and principles underlying the legal regime in question."[48] The law structures the factual inquiry.

Once we understand the foreseeability determination as the combination of these two moments of subjective decisionmaking, we can understand its relationship to abstraction choices. Imagine a lawsuit resulting from an autonomous vehicle that crashes when a firework goes off nearby that the computer mistakes for a green light, resulting in an injury to a passenger.

Let's think about the relevant abstractions. The full abstraction which specifies an autonomous vehicle will be enormously complex, with many inputs—training data, environmental sensors, GPS data, maps, and a destination—and lots of testing that was done which will shape the specification of the outputs. An abstraction can be very complex and still useful; perhaps we need to know a hundred or so pieces of information or pages of documentation to fully specify the inputs, outputs and relationship between them (which will include performance and safety metrics). Such an abstraction is nonetheless useful because once we have that, we need not know about the internal operation, which is infinitely more complex. Importantly to this hypothetical products liability case, we can imagine two competing abstractions that are relevant: 1) an autonomous vehicle that performs to some safety specification, or 2) an autonomous vehicle that performs to the same safety specification except with increased guarantees or

[48] Balganesh *supra* note 28, at 1592.

accuracy on traffic light corner cases. In this case, the first abstraction—whatever safety specification the car was made with—will be what was chosen by the engineers—and what is presented in court. The second would be a counterfactual that the court could consider if it chooses. In reality, there are infinitely many hypothetical counterfactual abstraction choices that could have been made, but it will usually suffice to compare the most relevant counterfactual.

The choice between these two abstractions sounds just like the framing step of a foreseeability inquiry. If asked to consider whether the event was foreseeable, a jury could be deciding between descriptions of the facts as "an autonomous vehicle that crashed after misreading a traffic light" or "an autonomous vehicle trained with millions of images of traffic lights that a mistook a firework explosion for a green light." As above, these are two ways of describing the reference class and the event that point to different outcomes from a foreseeability standpoint.

Analytically, the major difference between the choice of worlds presented by different abstractions and the choice of frames for foreseeability is that the former was made prior to the accident in question and could have changed the facts on the ground, whereas a court is charged, after the fact, with deciding whether the defendant *should* have made such a change. But the similarity between these claims demonstrates the conflict between them. There were once different possible technological arrangements, and the court is presented only with the result of them. The court must therefore not accept the technology as a fact about the world, unalterable, and itself ask for itself how the technical arrangements—the abstractions—could have been different. Once the need to surface the abstraction choices is recognized, then the problem looks like any other foreseeability inquiry the court would have considered: just as frustrating and unpredictable in result, but with the decision in the hands of the correct legal actors.

Seeing the foreseeability question as resetting the abstraction boundary does not just permit a more exacting examination of foreseeability on a binary basis; it also informs the distribution of responsibilities between different actors, such as the users and the developers. An abstraction boundary sets the expectations of responsibility for both those inside and outside the boundary. Where there is a well-defined boundary, if the designer *is* responsible for an outcome, the user is not, and vice versa. If the designer of the coffeemaker is not responsible for the grind (and therefore does not provide a built-in grinder), the user knows that she is. That choice of abstraction boundary is a well-defined choice. This corresponds well to a machine learning system that is purchased off the shelf. But in some algorithmic systems, a user may work with a designer to customize the problem definition, what data is used to train it, or other choices that need to be made. With such back and forth, there is not a well-defined or respected abstraction boundary between the system and its application, and it would be inappropriate to treat the user and designer as separate. Here, there is more likely to be overlapping fault, or at the least, it would be harder for a user or designer to say that the others' choices were unforeseeable. But seeing the question in terms of how the abstractions are chosen can clarify nonetheless.

Notably, our aim here is to point out that courts have the capacity to litigate claims about machine learning systems as they do any other fact pattern. In products liability lawsuits, judges are quite accustomed to asking whether a given danger of a product was foreseeable or could have been foreseen with more testing. The difference with machine learning and AI is that unlike technologies that obey relatively stable and well-understood laws of physics, the choices in the design of the technology result in much more variety in its composition and behavior. But by recognizing how these choices shape the technology, we can reduce the problem to the same sorts of inquires that are well-known to courts.

B. Abstraction Choices and Anti-Discrimination Law

The most commonly cited AI harms relate to bias, fairness, or discrimination. By now it is pretty well understood that choices in problem definition, training data, or other aspects of the machine learning problem can result in differential outcomes on protected classes.[49] But there is disagreement about whether anti-discrimination law can address these discriminatory harms.[50] The core of that disagreement is largely about the extent to which a discrimination defendant can buy a system off the shelf and use the black-box nature of the item as a shield against liability. If the discrimination defendant is responsible to look under the hood, then they can, in theory, be held liable, but if not then it is much more difficult to hold them either responsible or liable. We argue that this can be understood as a problem of abstraction boundaries.

In the United States, discrimination law canonically has two halves. "Disparate treatment" is generally taken to mean intentional discrimination, and "disparate impact" doctrine is purportedly effects-based, though perhaps more accurately described as effects-triggered.[51] Our concern here is

[49] Solon Barocas & Andrew D. Selbst, *Big Data's Disparate Impact*, 104 CALIF. L. REV. 671, 693, Pauline T. Kim, *Data-Driven Discrimination at Work*, 58 WM. & MARY L. REV. 857 (2017).

[50] Barocas & Selbst, *supra* note 49; Kim, *supra* note 49; Deborah Hellman, *Measuring Algorithmic Fairness*, 106 VA. L. REV. 811 (2020); Ifeoma Ajunwa, *The Paradox of Automation As Anti-Bias Intervention*, 41 CARDOZO L. REV. 1671 (2020).

[51] [[cite]]

primarily with disparate impact, because that is the kind of discrimination claim most likely to be generated by the use of machine learning.[52]

Disparate impact today involves a three-step burden-shifting scheme.[53] First, a plaintiff must point to an action the defendant took that had a disproportionate impact on members of a protected class.[54] Next, the defendant can respond by arguing that it is a "business necessity." The business necessity test is essentially the heart of disparate impact. Oversimplifying greatly, it essentially asks whether the disproportionate impact is justified by some business reason.[55] In employment, the standard is generally a low one: whether the challenged practice is job-related.[56] The third step in the analysis allows a plaintiff to demonstrate that there was a less discriminatory alternative that the defendant refused to use.[57]

The case of algorithmic discrimination arises where a person—say an employer—must make an allocative decision such as promotion or hiring, and decides to use a machine learning tool to predict the best outcome among candidates. The tool is advertised as the best available predictor of future job outcomes among candidates. Thus, when it evinces a disparate impact, triggering the three-part test, the defendant has a relativity easy time satisfying the business necessity defense—surely choosing a tool that would give him the best performance outcome is job-related.[58] The last part of the test, however, asks whether there was a less discriminatory means to do what the employer wanted. This is where the abstraction boundary comes in. The easiest way to imagine a similarly successful, but less discriminatory tool, would be to fix the model. But if we start from the position that the model is an unchangeable black box, that option is taken off the table.[59]

To understand this more conceptually, it will be helpful to zoom out in our understanding of what anti-discrimination law is trying to accomplish. The way anti-discrimination law is canonically understood is that there can be no discrimination without a discriminator.[60] But that limited view of what discrimination is does not fit with the common understandings of

[52] Barocas & Selbst, *supra* note 49, at 693.

[53] 42 U.S.C. § 2000e-2(k)(1)(A).

[54] *Id.*

[55] *Id.*

[56] Barocas & Selbst, *Big Data's Disparate Impact*, at 705.

[57] *Id.*

[58] *Id.*

[59] *See* Kim, *supra* note 49, at __.

[60] *See* Alan David Freeman, *Legitimizing Racial Discrimination through Antidiscrimination law: A Critical Review of Supreme Court Doctrine*, 62 Minn. L. Rev. 1049 (1978) (discussing the "perpetrator perspective"); Samuel R. Bagenstos, *The Structural Turn and the Limits of Antidiscrimination Law*, 94 Cal. L. Rev. 1, 42–45 (2006); [[others]]

discrimination outside of the doctrine.[61] Instead, today, most people recognize that there are forms of institutional and structural discrimination that cannot be traced back to a single action, but instead reflect the effects of compounded historical animus and decades of discriminatory policies.[62] With this understanding, anti-discrimination law as it is currently constituted does not and cannot address discrimination writ large; rather, each individual suit addresses only that subset of discriminatory harm fairly attributable to the defendant in the particular lawsuit.

This can be seen in the structure of the doctrine too. Functionally, while the first step of the three-step test is a strict liability, effect-based trigger, the business necessity and alternative practice tests reintroduce fault by asking if the defendant was justified in the action despite the discriminatory effects. But while a defendant is only liable if they are responsible for the discrimination, the lack of liability does not actually mean no harm occurred; rather a discriminatory harm is what triggered the lawsuit in the first place.[63] Thus, another way of understanding this is to separate out the discriminatory harm, and understand the function of anti-discrimination law to determine whether it is the fault of the defendant, the fault of someone else, or an accident. This sounds like a tort analysis, where we can usually agree there was an injury, and a court must attribute fault.[64] Indeed, Title VII is frequently described as a "statutory tort,"[65] and some consider it a "contemporary extension[] of tort law."[66] While the direct comparison to tort

[61] EDUARDO BONILLA-SILVA, RACISM WITHOUT RACISTS: COLOR-BLIND RACISM AND THE PERSISTENCE OF RACIAL INEQUALITY IN AMERICA (2003)

[62] [[cite]]

[63] *See* Noah Zatz, *Disparate Impact and the Unity of Equality Law*, 97 B.U. L. REV. 1357, 1375 at __ (arguing that we can categorize the discrimination injury as "status causation" and separate that analysis from responsibility for it).

[64] David Oppenheimer famously argued that in operation, despite being seen as divided between intentional and strict liability standards, anti-discrimination law actually has a great deal in common with negligence. David Benjamin Oppenheimer, *Negligent Discrimination*, 141 U. Pa. L. Rev. 899 (1993); *see also* Sandra F. Sperino, *Rethinking Discrimination Law*, 110 Mich. L. Rev. 69, 99 (2011) (referring to the least discriminatory alternative test as negligence). More recently, Stephanie Bornstein has argued for the recognition of a recklessness standard to apply to disparate treatment doctrine, which has its shown challenges with respect to understanding levels of culpability. Stephanie Bornstein, *Reckless Discrimination* (2017).

[65] *See* Charles A. Sullivan, *Tortifying Employment Discrimination*, 92 B.U. L. REV. 1431, 1432 (2012).

[66] John C.P. Goldberg & Benjamin C. Zipursky, *Torts as Wrongs*, 88 TEX. L. REV. 917, 918, 919 (2010); Sandra F. Sperino, *Discrimination Statutes, the Common Law, and Proximate Cause*, 2013 U. ILL. L. REV. 1, 35 (agreeing with Goldberg & Zipursky, *supra*, as a general matter, while arguing that courts should not port tort doctrine wholesale into employment discrimination law).

law is controversial,[67] one need not believe that disparate impact law has exactly the structure of a particular tort to understand the latter two steps as reintroducing a fault-based understanding of liability.

This brings us back to abstractions. When the defendant raises a business necessity defense by saying that he bought a machine learning tool that was supposed to be the most accurate solution for his business, he holds up the tool as a complete object; he is relying on the abstraction choices made by developers of the tool. Recall that the very purpose of formal abstraction is to tell a user of a tool that they need not worry about what's going on inside. In raising such a defense, the discrimination defendant is relying on this same idea, perhaps implicitly. It's not a trick or bad faith—an employer will often not have had the technical expertise necessary to understand and critique the tool, expecting that they can trust what they buy off the shelf.

If at a high level, the court's role is to determine whether the defendant was in fact culpable for the discrimination, then the court's role in the case of algorithmic discrimination is to decide whether a black box defense is good enough. Scholars have noted different reasons it should not be. Pauline Kim has argued that discriminatory correlations in a model might be spurious,[68] James Grimmelmann and Daniel Westreich argued that correlations might have no explanation other than relationship to protected class and should be rejected on that basis,[69] and one of us (Selbst) and Solon Barocas have argued that systems that are not based on human-understandable statistical relationships could be rejected unless there is documentation to justify the decisions in building them.[70] There are many reasons to think we can do better than just defer to the availability of the off-the-shelf system. But the lack of technical sophistication of many employers who are increasingly pressured to purchase ready-made technological solutions offers at least some reason to think we should take such a defense seriously.

The question of whether we should permit the black-box defense is therefore open, and once again requires understanding how the boundaries of the abstraction influence the apparent lines of responsibility. If the model is taken as given, then the only relevant questions become whether the

[67] *See* Charles A. Sullivan, *Tortifying Employment Discrimination*, 92 B.U. L. REV. 1431, 1432 (2012); Cardi, *The Role of Negligence Duty Analysis in Employment Discrimination Cases*; William R. Corbett, *What Is Troubling About the Tortification of Employment Discrimination Law?*, 75 Ohio St. L.J. 1027, 1032 (2014); Sandra F. Sperino, *The Tort Label*, 66 FLA. L. REV. 1051 (2014)

[68] Kim, *supra* note 49.

[69] James Grimmelmann & Daniel Westreich, *Incomprehensible Discrimination*, 7 CAL. L. REV. ONLINE 164 (2017).

[70] Andrew D. Selbst & Solon Barocas, *The Intuitive Appeal of Explainable Machines*, 87 FORDHAM L. REV. 1085 (2018).

defendant has the ability to comparison-shop between off-the-shelf AI products with different discriminatory responses. If the differences are not obvious, then this would seem to absolve the employer. But if we once again examine the abstraction choices made by the developer, we can ask how the inputs are defined—do they require a certain demographic distribution of data or a certain similarity to the data the model was trained on? That testing should be done, but if the abstraction accepts "any data," then the responsibility for ensuring applicability to a new demographic distribution is internal to the device, and the responsibility of the technology company; if only certain types of datasets are allowable inputs, then knowing that becomes the responsibility of the user. The decision about where the responsibility *should* lie is likely fact and context dependent. Perhaps if the system is packaged and marketed as off-the-shelf and complete, then the technology company bears the responsibility, but perhaps if it is a more bespoke solution, with the employer involved in the development, then they bear more blame. Whatever the outcome, it is surely a decision that belongs to the court, not to the technology company.

As a final note, there is a consequential difference between the abstractions in tort law and discrimination law from a plaintiff's standpoint. In tort law, a plaintiff can sue both the user or the creator of technologies in negligence and product liability respectively. Thus, the stance of the court case will often be that *someone* is responsible for the injury and that the plaintiff deserves compensation, but the court is left to determine who it is. In practice, it will still go to a jury and the answer can be that an injury was a blameless accident, but the abstraction boundary is more about distribution of liability. In discrimination law, there is no products liability equivalent. The decisionmaker is on the hook or no one. As a result, arbitrating the correct choice of abstraction boundary will usually be the difference between whether there is any finding of liability or not—and because of the formal entanglement of liability with the question of whether discrimination occurred at all, the rhetorical treatment of a no-liability judgment will be that no discrimination occurred at all.

C. Abstraction Choices and Factfinding

[This part is still to come. The idea is that the black box in, say, criminal proceedings, cannot just be accepted as a fact, and if the court fails to interrogate the abstraction choices, it will end up accepting certain facts as established without the requisite basis for judicial notice.]

III. How Abstraction Choices Inform Legal Outcomes

Once courts decide to resurface and examine the abstraction choices in the software they review, how does this help? Recognition of the abstraction choices let them take more evidence into account about proper practice in the production of machine learning. Experts can testify about how systems can be used, and typical practices in how they are built. Emerging standards within the industry can be used to inform factual scenarios in ways that would have been harder to analyze with a focus only on the system as given. In this Part, we work through three hypothetical examples—a tumor detection AI, a hiring algorithm, and [a third not yet worked out]—to demonstrate how courts will benefit from this analytical frame.

A. Case Study: Tumor Detection AI Gone Wrong

Consider a tort lawsuit that results from a physician's use of computer vision software developed with deep learning to assist in cancer diagnosis. The software misclassifies a tumor as unlikely to be cancerous, and the diagnosing physician follows this recommendation, causing the cancer to remain untreated for some amount of time, killing the patient. The patient (through her executor) could potentially bring a lawsuit against the doctor or hospital for negligence (medical malpractice) or against the developers of the software under a products liability theory. The court will be tasked with deciding which of these defendants, if any, is liable for the injury.[71]

In such lawsuit, the different defendants would likely all claim they were reasonable, and perhaps they would point fingers at each other. The doctor would argue that he was using the equipment properly, the hospital that they adequately trained the doctor on the new equipment, and the developer would argue that the error was probably user error. The doctor might even argue that the particular outcomes were unforeseeable in some circumstances. Depending on the facts, each defendant could theoretically admit some portion of fault, making the case easier for the court, but for sake of illustration let's assume they each deny it.

There are many different ways that such a system can go wrong, and we discuss a few scenarios below. Let's start with a simple example. Suppose that the error in diagnosis occurred because the supervised learning model powering the software was overfit to the training data. This is as straightforward a failure as one can get in machine learning. It is basic

[71] For convenience of discussion, we collapse the roles of judge and jury in this Section. If an issue is for the judge then the discussion applies straightforwardly, but if an issue is reserved for the jury, then the discussion will end up applying to jury instructions instead. It doesn't change the points we make.

knowledge in the field that just because a model training procedure fits the training data well does not mean it will fit the unseen future data well.[72] This is due to a mathematical phenomenon known as the bias-variance tradeoff: roughly speaking, if a model is too sensitive to the training data, it introduces error due to variance, and if a model is too inflexible it introduces error due to bias.[73] Because a very sensitive model may perform very well on the data it was trained on, but very poorly on previously unseen cases, the best way to estimate a model training procedure's performance on the "real" data is generally to set aside some of the available training data into test data, run the training procedure on the remaining data, then see how the model performs on unseen data. Skipping this step and only reporting how the model performs on training data as an estimation of how well the model might perform in practice would be a cardinal sin in the field. Because it is such a basic principle, this initial example is rather unlikely to occur in practice, but it is useful for demonstration.

It is important to note that machine learning systems operate on probabilities. A system works well when it predicts a certain percentage of cases correctly; it cannot get all of them. Accordingly, when a patient is misdiagnosed, the fact of an error does not imply that the doctor messed up or that the machine is unreasonably dangerous. If a machine predicts a high percentage of cases correctly overall, then it will likely be considered quite safe at first glance.[74] Here, the machine will have a high error rate in practice. While it is impossible to say how high an error rate is "too" high, we can stipulate that the product is defective.

In a products liability context, the court's role is explicitly to question the technology, so a court will naturally question how it was designed, and there is little difficulty. But consider a medical malpractice suit against the doctor, where the technology company is not a defendant. In this case, the court seems less likely to ask the question of whether the technology was designed defectively. Product defects are not part of the prima facie negligence case. Instead, the court would ask whether the doctor behaved reasonably by relying on a machine learning technology that is billed as being more accurate than most doctors. It is hard to imagine that this is unreasonable behavior, if the technology is considered a fact that already exists.[75]

But consider the abstraction that defines the model in this instance. The tool is built to take in a certain input—image data—and output a

[72] Murphy, Kevin P. *Machine learning: a probabilistic perspective*. MIT press, 2012. Page 22

[73] JAMES, *supra* note 11, at 34.

[74] A. Michael Froomkin et. al., *When AIs Outperform Doctors: Confronting the Challenges of A Tort-Induced over-Reliance on Machine Learning*, 61 Ariz. L. Rev. 33, 33 (2019).

[75] *See* Selbst, *supra* note 25, at __.

decision on whether a tumor is present in that image. Recall that an abstraction is composed of well-defined inputs and outputs and the function that transforms inputs to outputs. Importantly, the function here was only specified to perform well on its own training data, not real-world data it might encounter. What is the abstraction choice communicating about lines of responsibility? Because the abstraction of the model does not include an ability to generalize, the technology is communicating that it is the user's responsibility to ensure that generalization works; anything on the outside is the responsibility of the user. This should complicate the court's analysis. Though the technology *company* is not a party, the technology itself can almost be seen as testifying as to its own view of responsibility. The court must then consider who, according to standard principles of negligence, should be responsible for this particular error. The court should want to hear expert testimony on the creation of the machine and the expectations of the users. In this case, any expert would testify that this is not the way anyone in the computer vision field chooses to specify a problem; in order to even be a proper a classification model, it must meet some specification of generalizability. Thus, looking at the abstraction demonstrates that this is clearly a defective product.[76]

Here, the court could use this this information in one of two ways: 1) The court recognizes that the doctor and hospital chose to use a technology that assigned responsibility for this issue to the user, and therefore assigns liability to them, or 2) The court recognizes that abstraction choice, and nonetheless lets the doctor off the hook, because it is not a requirement for reasonableness that the doctor understand the technical details that were not advertised. Either one of these directions is defensible in some sense, even for a product as clearly defective as this. The important point that we want to make is that the court must recognize that the technology itself is making such a claim and could have been designed differently.

Now consider a more plausible scenario: perhaps the algorithm's generalization performance was estimated to be high, but certain conditions caused it to behave worse than expected. For instance, the training data used to develop the model and estimate generalization performance was of very well-lit images, but the doctor or hospital tried to use the software on a low-quality image. In general, if a supervised model is deployed on data drawn from the same statistical distribution as the validation data, generalization error estimates will be similar in deployment; if it is not, the best model for

[76] Given the statistical nature of the question, the plaintiff might still run into trouble proving causation, but causation asks whether the defect is more likely than not the cause of the injury, and for a defect as basic as this, we imagine that most juries would find that to be true.

the test data may be based on spurious or unhelpful patterns which do not translate to the real world.[77]

So whose responsibility was it to make sure that the images were well-lit? Consider two ways that the inputs could have been defined: Either the model took in "images" or "well-lit images." If the input was defined as well-lit images, that implies that a poorly-lit image is not an acceptable input to the model, meaning that the lighting is the user's responsibility. If the input is defined as just any old image, then that suggests that the technology should work on any image, and it is the developer's responsibility to either train on poor light or have software that automatically compensates for the lighting before processing.

By the time a court encounters the injured patient, the technology has been set; either the proper input was "images" or "well-lit images." The abstraction choice was made long ago. In litigation, the doctor will point to the tech company, and vice versa. If the court accepts the version of the technology that exists, without questioning, then the defendant who was at fault will probably become clear. But this will have been a choice dictated by the developer of the technology, for reasons pertaining to the convenience or profit of the company, not the normative concerns underlying tort doctrine. Perhaps it is better for the company's sales to say their technology can handle images of differing lighting quality—or more likely say nothing at all about it, leaving it as an implication—but it is overall safer for the hospitals to ensure that images are well lit. Because safety, not preservation corporate profits, is the primary goal of tort law, in that case the court should be assigning liability to the hospital or doctor, not the technology company—irrespective of how the technology was designed in this instance. That is to say, the definition of the technology's inputs—part of the model abstraction—would assign responsibility one way or another, but the court should decide for itself, and to do so, it must recognize that the abstraction choice could have been different; ensuring good lighting could have been set outside the system.

If the doctor claims that the error was unforeseeable, and therefore that he did not breach his duty of care, the court should also question what work abstraction boundaries are doing. In this case, the doctor would be saying that the technology accepted all images, not just well-lit images, so there was no way for him to know that poorly-lit images would cause a misdiagnosis. If the doctor's factual claim about the technology is accurate, and the inputs were all images, then the technology would essentially be agreeing with him; the developers' choice to deal with lighting internally *means* that the doctor should not have to know about challenges with lighting issues. But if the court were to challenge this assignment of responsibility for

[77] Caruana et al., 2015; Arjovsky et al., 2019; Ilyas et al., 2019

knowledge about possible errors, the court might seek evidence about whether certain kinds of failures are common and should be expected, such that irrespective of the technologists' choices, doctors should know to look out for them. The technology should not dictate either ultimate responsibility or foreseeability for outcomes.

In one final permutation, consider a model that performs systematically poorly on a certain subpopulation that the patient in question was a part of, such as a racial group. Machine learning models often perform poorly on already-disadvantaged groups.[78] These failures have increasingly become high-profile in recent years. A notable example of this is described in the ProPublica article Machine Bias, in which journalists found an algorithm predicting recidivism risk was "particularly likely to falsely flag black defendants as future criminals, wrongly labeling them this way at almost twice the rate as white defendants," while "white defendants were mislabeled as low risk more often than black defendants."[79] The story of the influential researchers who found evidence of systematically poor performance of computer vision models on black women were recently covered in the popular documentary Coded Bias.[80]

Like the prior hypotheticals, the abstractions here can be described in terms that either include or exclude the demographic generalization issue. But note what the equivalent "statement" by the technology is here, compared to the lighting scenario. A system could be meant to work only on well-lit images, but here, would it be acceptable to claim that the software is only meant to work on lighter-skinned individuals? Probably not. The normative concerns here have a different valence than those in the lighting scenario. But that also does not clearly dictate the outcome in a medical malpractice suit. If the machine does not work on light-skinned individuals, the company will have plenty of incentive not to admit it, and not to say so in the documentation, because society does not think it acceptable. But from a safety standpoint, a court might find it important to hold doctors liable for diagnosing a person with a machine that will likely not work for them, to make sure that doctors know whether the tools they use are appropriate for the task at hand. Again, the important thing is that it is the court's role to surface the hidden choices, and then decide.

[78] Nicol, Casey, and MacFarlane 2002; Rodger and Pendharkar 2004; Tatman 2017; Buolamwini and Gebru 2018; Bolukbasi et al. 2016; DeVries et al. 2019; Angwin et al. 2016; Raji and Buolamwini 2019; Koenecke et al. 2020; Obermeyer et al. 2019

[79] Julia Angwin et al., *Machine Bias* PROPUBLICA (May 23, 2016)

[80] Coded Bias

B. Case Study: Discriminatory Promotion AI

Suppose a company is using an automated system to predict likelihood of good performance if an employee is promoted. While using the model, the company passed over a female applicant for a promotion. The algorithm is found to have disproportionately suggested men for promotion at the defendant's organization. The female applicant sues the company under a disparate impact theory.

In response to the suit, the defendant will raise a business necessity defense, arguing that the use of the tool was job-related. According to the EEOC's Uniform Guidelines on Employee Selection Procedures, widely recognized as the presumptive standard for validation, a quantitative test such as this must be validated according to one of three metrics, else it is presumptively considered not job related.[81] The relevant metric here is "criterion validity," "consist[ing] of empirical data demonstrating that the selection procedure is predictive of or significantly correlated with important elements of job performance."[82] As a statistical method, a competently created machine learning tool will generally satisfy this validation, and accordingly the business necessity test.[83] The burden will then shift to the plaintiff to demonstrate that there was an "alternative employment practice" that was less discriminatory and the employer refused to use—here a different version of the model.[84]

As a practical matter, most employers will not have the technical capacity to build and train such a model themselves. Thus, in most cases they will purchase the software, either off the shelf, as an existing package, or as a bespoke model, co-designed with a technology company. They purchase the technology, which probably *is* more predictive of their future job outcomes than other methods they would use to promote employees, and ipso facto, establish business necessity.

Once business necessity is established, the plaintiff bears the burden of proof to demonstrate a lack of an alternative. This is where the abstraction boundaries start to matter. In the case where companies buy off the shelf, it seems natural to think of the product as a finished good, a take-it-or-leave-it box with the only options being to use it or not. If a plaintiff wants to prove the existence of an alternative, she must show that there is another piece of software in the market that would have done as well for the company but be less discriminatory; there is not really a way to say that the employer should

[81] Uniform Guidelines on Employment Selection Procedures, 29 C.F.R. §§ 1607.3–1607.5

[82] *Id.* § 1607.5(B).

[83] Barocas & Selbst, *supra* note 49, at 709.

[84] 42 U.S.C. § 2000e-2(k)(1)(A)

have changed the model. Then, because is not clear that such a competitor will exist, and courts tend to be deferential to employers regarding what is best for the business,[85] this showing will likely fail.

Note the work that the abstraction boundary and choices are doing here. Because the system is off-the-shelf, there is a boundary between the creation of the system and the use of it. It therefore seems natural to suggest that the defendant wasn't in position to do anything. But consider an alternative similar to the biased medical imaging model above—suppose the software company offered documentation saying that their software works best on certain specified demographic distributions. This may not appear likely because no software company wants to admit their technology is biased by race or gender, but there is also a push within the technical community for increased documentation as to the limits of data or models.[86] Regardless, let us just imagine such a possibility. This would change the model abstraction—now the model takes as inputs only data that matches the distribution they documented; the requirement to ensure proper use of the tool—that the population tested matched the tool's training data—now rests with the user of the tool. Once again, a different technical configuration offers different lines of responsibility.

Pauline Kim has argued that employers who use machine learning models should bear the burden to demonstrate more than just statistical validity, but relevance to job performance in some deeper sense.[87] If we understand the combination of business necessity and alternative employment practices to combine to a rough fault-based test, then these different possible abstraction choices can help support Professor Kim's argument. The defendant comes to court with a single version of the technology, which is in essence making claims about the distribution of responsibility. The technology could have been created differently and could be "arguing" for a different distribution of responsibility. But the vision proffered by the technology itself should not be the determinant; it's up to the court.

[85] Kim, *supra* note 49, at 908

[86] Researchers have proposed standardized documentation practices including datasheets that describe the sources and known biases in a dataset used for training a model, and model cards that describe the "operating characteristics" of a model such as its accuracy, ability to generalize, known biases and similar characteristics. *See* Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III & Kate Crawford, *Datasheets for Datasets*, Proc. 5th Workshop On Fairness Accountability & Transparency Machine Learning 1, 1 (2018); Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji & Timnit Gebru, *Model Cards for Model Reporting*, Proc. Conf. On Fairness Accountability & Transparency 220, 220 (2019); Data Nutrition Label, IBM AI Fact Sheet, etc.

[87] Kim, *supra* note 49, at __ 921.

Professor Kim specifically argues that defendants, not the plaintiffs, should bear the burden to demonstrate that the technology is not overly discriminatory.[88] One way to think about the importance of burden shifting is that when the defendants make their business necessity defense, they are not just making factual claims about their practices and then shifting the burden to the plaintiffs; rather they are also making factual claims about what counts as the device. Recall the facial recognition example in Part I, with three products—amusement park access, bank authentication, and TSA-verification. The same thing could apply here. Imagine a company called HireZone that has two products: "Los Angeles" and "Salt Lake City". We wouldn't necessarily think of them as the same type of system at all. If a Los Angeles startup said "we used the Salt Lake City product because it was cheaper and it's basically the same thing," then the court would expect some factual development, perhaps expert evidence, to establish that claim. The court would not just assume that one of the parties was entitled to make a factual assertion like that. When a company comes in with any type of technology, they are making a factual claim as to what counts as the boundary of that technology, and the court should, where appropriate, assign the burden to them to prove their factual claim, that the technology ends where they say, and that they should not be held responsible for taking the context into account.

This all becomes much more obvious when considering the scenario where the employer co-develops the software with the technology company. While the end-product may be the same, the lines of responsibility are laid out differently by the design process of the technology. The employer had a hand in some of the internal choices, and therefore the abstraction boundary still points to the employer as responsible. Accordingly, no employer in this circumstance could plausibly argue that there was no alternative to the technology that resulted by simply pointing to the object. They might still be able to argue it was the best option, but they will have to do more work, or at least the plaintiff's burden will be easier.

In sum, the blame for the discriminatory outcome of data-driven hiring could properly lie with the employer, the software company, both or neither. Because the technology company is not part of the lawsuit and the employer will often lack technical expertise, it will often be tempting to treat the technology as a fixed object, and to hold that the employer only had the choice to use it or not. Courts should resist automatically treating the technology that way, and interrogate the alternatives that could have existed. Maybe the defendant employer really is not at fault—maybe they shopped around or the technology was advertised as less discriminatory than it turned

[88] *Id.*

out to be—in those cases, the defendant would rightfully be off the hook, but the decision would have been made by the court, not the technologists who build the product.

C. Case Study: [Something to demonstrate the epistemic issue]

[TK]

CONCLUSION

[TK]