ABSTRACT
Black-box heterogeneous treatment effect (HTE) models are increasing-ly being used to create personalized policies that assign individuals to their optimal treatments. However, they are difficult to understand, and can be burdensome to maintain in a production environment. In this paper, we present a scalable, interpretable personalized experimentation system, implemented and deployed in production at Meta. The system works in a multiple treatment, multiple outcome setting typical at Meta to: (1) learn explanations for black-box HTE models; (2) generate interpretable personalized policies. We evaluate the methods used in the system on publicly available data and Meta use cases, and discuss lessons learnt during the development of the system.

CCS CONCEPTS
• Information systems → Personalization.

KEYWORDS
heterogeneous treatment effects, personalization, interpretability

1 INTRODUCTION
In conventional A/B testing, individuals are randomly assigned to treatment groups, with the goal of selecting a single best treatment (on average) for the entire population. Increasingly, internet companies are taking a personalized approach, making use of heterogeneous treatment effects models (HTE) that predict individual-level treatment effects for each individual, and personalized policies that aim to deliver the best treatment to each individual [16, 17].

Current state-of-the-art HTE models used for personalization frequently leverage black-box, un-interpretable base learners such as gradient boosted trees and neural networks [29, 52]. Some systems even generate HTE predictions by combining outputs from multiple models via ensembling or meta-learning (e.g., stacking, weighting) [31, 39, 42]. Moreover, the complete end-to-end personalization system (from individual features to treatment decisions) sometimes uses these treatment effect predictions as inputs to additional black-box policy learning models [24].

There are several challenges with deploying these black-box approaches at internet companies. First, black-box HTE models and resulting policies are difficult to interpret, which can deter their uptake. Second, as dataset size and the number of features increases, which is common in internet companies, the burden of maintaining such models in production increases [45]. Although there exists work that addresses these challenges via optimal interpretable policy learning methods [1, 57], we have found these methods computationally intractable on large-scale datasets.
In this paper we present a scalable, interpretable personalized experimentation system, implemented and deployed in production at Meta. The system is divided into a HTE model interpretation stage and an interpretable policy generation stage. Our HTE model interpretation approach is responsible for helping product teams understand how black-box HTE models are performing personalization (deciding which treatment group to assign to each user). After that, our interpretable policy generation approach generates segments of individuals to recommend different treatment groups to, to capture potential gains from personalization. A product team then has the choice of deploying a black-box policy (implied by the black-box HTE model) or an interpretable policy generated by our methods. Figure 1 shows the system.

2 RELATED WORK

**Large-scale HTE personalization.** Many different types of HTE models have been proposed (see previous section), and different aspects of utilizing them for personalization in industry have been considered. For example, Lada et al. combined experimental and observational data to estimate individuals’ heterogeneous response to page recommendations to personalize Facebook News Feed [32]. Xie et al. tackled the problem of false discovery, the chances of which increases with large-scale data [55]. The focus of this paper is on increasing understanding of black-box HTE models, an area that has received less attention, and deployment of interpretable policies in lieu of black-box policies.

**Segment discovery.** Our work surfacing segments with HTE produces a representation analogous to segment discovery methods. These methods are divided into: (1) statistical tests to confirm or deny a set of pre-defined hypotheses about segments [2, 48]; (2) methods that aim to discover segments directly from data [9, 23, 40, 53]; (3) methods that act on predicted individual-level treatment effects to discover segments [12, 14, 34]. However, the comparison of segment finding algorithms is still an open question. Loh et al. empirically compared 13 different segment-finding algorithms [36], finding that no one algorithm satisfied all the desired properties of such algorithms. Closest to Distill-HTE, the method we use in our system, are [14, 34]; we compare these three methods in Sec. 5.1.

**Policy learning.** Many methods have been proposed to learn policies from experimental or observational data. However, most methods produce black-box policies [28, 30, 43, 49, 56]. Several existing works [1, 4, 25, 57] construct interpretable policies with similar representations (segments) as the methods we use in this paper, but do so by constructing exact optimal trees, which is NP-hard [33]. Other works relax this by constructing approximate solutions to the exact optimal trees [5, 11]. In practice, we found these exact or approximately exact methods prohibitively slow on large-scale data and unsuitable for a production environment (see Section 4.3.1 for runtime results when we tried these methods at Meta). With a focus on scalability, in this paper we consider methods using greedy heuristics to approximate optimal trees, similar to [5, 11, 19], methods using distillation techniques to approximate black-box HTE models, similar to [6, 37], and methods that directly learn policies from data, similar to [27]. We do not propose novel policy learning methods in this paper, instead focusing on describing the system we built that evaluates different methods on Meta use cases.

3 PROBLEM SETTING AND NOTATION

Our setting is that of large-scale randomized experiments, also called A/B tests, commonly conducted at internet companies to

![Figure 1: Left: Standard randomized experiment. During the experimentation phase, individuals are randomly assigned to treatments and control, and the best treatment is selected. At the deployment phase, that treatment is applied to all individuals. Right: Personalized experiment. At experimentation time we generate a policy that will decide, at deployment time, what treatment to use for each individual. Our contributions are depicted in the dashed red box: i) an approach to understand existing black-box HTE policies, to empower product teams to decide whether they can be trusted, and ii) an approach to generate interpretable policies, that can be used instead of the black-box policies.](image)
Figure 2: Distill-HTE: Multitask decision tree (MTDT) learned by distilling treatment effect predictions from a black-box HTE model. After an MTDT is learned (bottom left), we “unroll” the tree such that each terminal node is a bar (segment) in the barplot (bottom right). Red and green bars are respectively segments with negative or positive predicted treatment effects; gray segments are not statistically significant. In our system, feature splits defining each segment (e.g. clicks_7d > 10 and views_7d > 100) are overlaid on top of each red/gray/green bar and also appear when a product team user hovers over the bar. 

Heterogeneous Treatment Effects (HTE). Product teams are not solely interested in average treatment effects, but also HTE, i.e. effects of new product features on particular groupings of individuals that are significantly different from the average individual. Reasons include ensuring that new product features are not showing gains on certain individuals at the expense of other individuals, delivering the best product feature to each individual, etc. For each individual i, outcome j, and treatment group k compared to control, individual-level treatment effects are defined as $\hat{\tau}_{ij}^{(K)} = Y_{ij}^{(k)} - Y_{ij}^{(0)}$, where $Y_{ij}^{(0)} = 0$. T-Learners [3, 31], a type of HTE model, replace $Y_{ij}^{(k)}$ by $\hat{Y}_{ij}^{(k)}$, the plugin estimators of individual potential outcome surfaces. Throughout this paper, we use $\hat{T}_{ij}$ to refer to any black-box estimate of treatment effects, and $\hat{Y}_{ij}$ to refer to any black-box estimate of potential outcomes. Our system is agnostic to the specific choice of HTE model, as long as it outputs $\hat{T}_{ij}$ and $\hat{Y}_{ij}$.

Certain HTE models produce predictions not at the individual-level, but rather at the segment-level, i.e. a grouping of individuals. Denote a segment, $S$, to be a set of indices representing a group of individuals. In our system, the segments learnt do not overlap (i.e. each individual belongs to exactly one segment).

**Deterministic policies.** Policy $\Pi : \mathbb{R}^{p} \rightarrow \{0, 1, \ldots, K\}$ decides, for individual i with features $X_i$, which of the $K+1$ treatment groups to assign to the individual. In our system, we learn deterministic policies rather than random policies to ensure a more consistent UX experience for individuals. We evaluate different policies using offline policy evaluation (OPE). Concretely, let $G_k$ denote the set of individuals in the dataset whose treatment group matches that prescribed by policy $\Pi$. We compute:

$$\frac{1}{N} \sum_{i=1; i \in G_k}^{N} \frac{Y_i}{p_i}$$

where $p_i$ is the propensity score for individual i ($p_i = \frac{1}{K+1}$ in a randomized experiment with $K+1$ groups).
4 INTERPRETABLE PERSONALIZED EXPERIMENTATION

Our system performs interpretable personalized experimentation in two independent stages: black-box HTE model understanding, and interpretable policy generation. We now describe the two stages.

4.1 Understanding Black-Box Heterogeneous Treatment Effect Models

4.1.1 Considerations when designing the system. In developing a black-box HTE model understanding method for Meta, we had several considerations:

- The method should be a post-hoc method that can be applied to any type of HTE model, as the HTE models may be refreshed and retrained frequently.
- The method should handle multiple treatment groups and multiple outcomes, as is typical in industry settings.
- Since the outputs of the method will be placed in front of product teams to aid their understanding of the black-box HTE models, they should be understandable by engineers, data scientists, and product managers, with little training.

4.1.2 Method. We use multi-task decision trees (MTDT), which extend single-task decision trees by combining the prediction loss on each outcome, across multiple outcomes, into a single scalar loss function. This simple representation is effective and suitable for locating segments where individuals have heterogeneity across multiple outcomes. In the MTDT, each task (label) is the predicted treatment effect for an outcome, and each node is a segment identified to have elevated or depressed treatment effects across multiple outcomes. Since we learn these trees using distillation techniques, we call the method Distill-HTE. Figure 2 illustrates the method.

Learning objective. We suppose we have access to pairwise predicted treatment effects \( \hat{T}_{ij} \) from the black-box HTE model. We will learn one MTDT model for each treatment group compared to control. To learn an MTDT model in a HTE model-agnostic fashion, we leverage model distillation techniques, taking the HTE model as teacher and the MTDT model as student [7, 22]. Let \( \hat{F} \) be the prediction function of an MTDT model for treatment group \( k \) compared to control, using the following distillation training objective that minimizes the loss between predicted treatment effects \( \hat{T}_{ij} \) and \( \hat{F} \):

\[
L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{J} c_j \cdot l(\hat{F}_{ij}(X_i), \hat{T}_{ij}(k))
\]  

(3)

\( \hat{F}_{ij} \) is the MTDT model’s prediction for outcome \( j \). \( c_j \) is a weight that encodes how much outcome \( j \) contributes to the overall loss, and \( l \) is the loss function used to train model \( \hat{F} \) (mean squared error in this case). When \( c_j = 1 \), as our system assumes by default, the loss of each outcome contributes equally to the overall distillation loss. However, all outcomes still affect jointly the training of the model in a way that training one model for each outcome independently would not [8].

Improving robustness. We do the following: (1) Terminal nodes without sufficient overlap (i.e., enough treatment AND control points) are post-pruned from the tree, and predictions are regenerated; (2) Confidence intervals are provided for treatment effects within each node; (3) Honesty criterion [3]: splits and predictions are determined on different data splits.

Interpretable output. An MTDT can be visualized as a tree with each node having \( J \) predicted treatment effects, one per outcome, and edges being feature splits (Figure 2 bottom left). We interviewed product teams at Meta to determine the best way to present this (see Section 6.3 for interview findings), which led us to design the barplot-style output (Figure 2 bottom right) for product teams.

4.2 From Treatment Effects to Policies

When there are only two treatment groups and one outcome, it is easy to derive a policy from treatment effect predictions. For example, a policy derived from one of the two barplots in Fig. 2 bottom right could be: For the red segment where predicted treatment effect on engagement is negative for Treatment A vs. Control, assign to Control. For the green segment, assign to Treatment A. However, it is not as easy to derive policies from treatment effect predictions when there are more than two treatment groups or more than one outcome.

Multiple outcomes. If another outcome is added (e.g., latency in Figure 2), both outcomes have to be considered together to derive a policy. One way to tradeoff between multiple outcomes is to modify the weights \( (c_1, \ldots, c_J) \) used. This technique, from the multi-objective optimization literature [18] and commonly used for ranking models in industry [10, 15, 50, 51], weights and adds multiple outcomes to form a single outcome: \( \hat{Y}_{i}(\hat{W}_i) = c_1Y_{i1}(\hat{W}_i) + \cdots + c_JY_{ij}(\hat{W}_i) \).

In the second stage of our system, we combine predicted multiple outcomes from HTE models this way to generate policies. The weights \( (c_1, \ldots, c_J) \) can be the same weights used when training MTDT (Equation (3)), or provided by product teams, reflecting their preferences for tradeoffs (e.g., no more than x loss in latency to achieve y gain in the engagement). They could also be assumed to be 1 by default (equally weighted), or tuned online [35].

Multiple treatment groups. \( K + 1 \) treatment groups and \( J \) outcomes yields \( K \) MTDT models, with each model predicting \( J \) outcome treatment effects for each treatment group compared to control. Generating a single policy would require combining multiple MTDT, and it is not obvious the best way to do. Hence, while sufficient for understanding treatment effect predictions, MTDT is not suited for policy generation when there are more than two treatment groups. This motivates the second stage of our system: directly learning a single policy that can assign more than two treatment groups.

4.3 Interpretable Policy Generation

4.3.1 Considerations when designing the system. In developing interpretable policy generation methods for Meta, we had several considerations:

- Like the HTE model understanding method described in the previous section, the interpretable policy generation method should handle multiple treatment groups and multiple outcomes, and moreover, do so in one single model (the learned policy). Achieving this allows us to deploy only one policy in production, rather than multiple policies and then having to reconcile between them, which adds tech debt.
- Unlike the HTE model understanding method which is run immediately after the black-box HTE model is trained, product teams...
would like to repeatedly interact with and tune the number of segments, size of each segment, weights to trade off multiple outcomes, etc.

- The method must be able to handle large-scale datasets with millions of individuals and hundreds of features.

We first considered optimal tree-based policy learning methods, as reviewed in Section 2. However, in our initial experiments on a small sample of 100 thousand data points with 12 features and three treatment groups, [57] that solves an exact tree search problem took 2.5 hours to run. With a quadratic runtime in the number of data points, it was impossible to run on our datasets that sometimes exceed 100 million data points. Other methods that find approximate solutions to exact optimal trees using coordinate ascent should be faster but their software is proprietary [1, 5].

In the end, we implemented different methods in our system: (1) a method that approximates optimal tree finding methods with greedy heuristics (GreedyTreeSearch-HTE); (2) a method using distillation techniques to approximate black-box HTE models (Distill-Policy); (3) a method that directly learns policies from data (No-HTE); (4) a method that ensembles different interpretable policies while remaining interpretable (GUIDE). All methods produced personalized policies that we visualize as a sequence of rules (Figure 3 top). All of these methods were inspired by close counterparts in literature, but we implemented them in slightly different ways to better suit our setting. We review this in detail in the next section.

4.3.3 Methods that take HTE model predictions as input. The methods proposed in this section need an already trained HTE model from which predicted outcomes $\hat{Y}_i$ can be obtained.

**GreedyTreeSearch-HTE** directly solves the following optimization problem in the space of trees with pre-determined maximum tree depth: $\Pi^* = \arg\max \sum_{i=1}^{N} \psi_i(\Pi(X_i))$, assuming without loss of generality that higher outcomes are better. This can be treated as a cost-sensitive classification problem (NP-hard), where the cost of assigning a point to a group is the negative value of the predicted outcome [13]. To achieve a scalable implementation, we solve the optimization problem greedily instead of resorting to exact tree search over the whole space of possible trees, and obtain personalized policy $\Pi$. See Algorithm 1 in the Appendix for the implementation. Other methods have been proposed to approximately solve this problem, for example those used in [11]. However, the greedy heuristic we chose offers the most simplicity and scalability.

**Distill-Policy** starts from the naive policy $\Pi$ implied by the outcome predictions $\psi_i(0), \ldots, \psi_i(K)$, i.e. $\Pi_{HTE}(X_i) = \text{argmax}_{W} \psi_i(W)$. Then, we train a decision tree classifier to predict $\Pi_{HTE}(X_i)$ given $X_i$. We create a personalized policy from this tree by assigning all individuals in the same terminal node (segment) to the majority treatment group of individuals in that segment.

4.3.4 Ensemble methods. We can already generate interpretable tree-based policies using the methods described above. However, different policies may exhibit different strengths in different feature regions, and simply training trees with deeper depth does not necessarily improve the resulting policy. We leverage ensemble learning to identify such regions, with the hope of generating a better policy. Our motivation of combining different policies while remaining interpretable is shared by [51], but we do not use their method, because their method, being an intersection of segments from different trees, yields many more segments than suitable for our setting. While there exists other ways to ensemble policies, such as SuperLearner [39], they result in non-interpretable policies.

We ended up designing short trees, which we call guidance tree, that ensemble two policies with just one more feature split (see Figure 4 in the Appendix for an example). **GUIDE-OPE** finds this feature split using exhaustive search on the OPE criterion (Equation (2)) to find one optimal feature split that becomes the top of the guidance tree. **GUIDE-ExploreExploit** treats individual interpretable policies as "arms", as in the contextual bandits literature [47], and uses explore-exploit to learn another policy on top of it that selects the candidate policy. See the Appendix for implementation details.

5 RESULTS ON PUBLICLY AVAILABLE DATA

In this section, we evaluate the methods behind the interpretable personalized experimentation system on publicly available data.

5.1 Comparing Explanations of HTE Models

We compare the Distill-HTE method proposed in Section 4.1 against several other segment finding methods that take HTE model predictions as input: (1) Virtual Twins (VT) [14]; (2) R2P [34], a recent, state-of-the-art method. We also compare to a black-box HTE model: a T-Learner that does not find segments but rather provides...
one prediction per individual. We train T-Learners using gradient boosted decision trees (GBDT) and decision tree (DT) base learners.

**Setup:** The COVID dataset was used in [34] and uses patient features as in an initial clinical trial for the Remdesivir drug, but generates synthetic outcomes where the drug reduces the time to improvement for patients with a shorter period of time between symptom onset to starting the trial. We chose this dataset because it has a specific finding of heterogeneity. The second dataset, Synthetic A, was used in [3]. For R2P we used the implementation of R2P provided by the authors. For VT we used our own implementation, first training a black-box model (we use random forest (RF) and GBDT) to predict potential outcomes, then training a tree to explain the difference in potential outcomes.

**Evaluation:** Since these datasets have synthetically injected ground truth treatment effects, we can measure the accuracy of each method. We use the Precision in Estimation of Heterogeneous Effect (PEHE) metric [20]: 

\[
\begin{align*}
\text{PEHE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left( \left( Y_i^{(1)} - \hat{Y}_i^{(0)} \right) - \left( Y_i^{(1)} - \hat{Y}_i^{(0)} \right) \right)^2},
\end{align*}
\]

Unlike bias and RMSE computed relative to ground truth treatment effects, PEHE requires accurate estimation of both counterfactual and factual outcomes [26]. We also compute between- and within-segment variance. For each dataset we generate ten train-test splits, on which we compute the mean and standard deviation of estimates.

**Hypothesis:** The best segmentation methods should have low PEHE and produce segments with low within-segment variance, and high between-segment variance.

**Results:** Table 1 presents the results. We make a few observations: (1) **Black-box vs. white-box:** As expected, GBDT T-Learners perform well as they do not have interpretability constraints, unlike all the other methods (VT, Distill-HTE, R2P), all of which modify standard decision trees while still remaining visualizable as a tree. Yet, Distill-HTE tends to be far more accurate than R2P, in terms of PEHE. (2) **Optimization criterion:** R2P, the only method of those presented here, that considers not only homogeneity within segments but also heterogeneity between segments, has the highest between-segment variance. Other methods that do not try to increase heterogeneity between segments do not fare so well on this metric. However, R2P does this at the expense of PEHE. (3) Impact of distillation: While the T-Learner DT model did not perform well, being worst in terms of PEHE on all datasets, Virtual Twins RF, Virtual Twins GBDT and Distill-HTE that train modified decision trees have a marked improvement over T-Learner DT, suggesting that distilling a complex GBDT or RF teacher rather than learning a tree directly is beneficial, which agrees with the existing distillation literature. (4) **HTE model class:** The choice of HTE model matters, with Virtual Twins not performing as well when using an RF HTE model compared to a GBDT HTE model. Similarly, GBDT T-Learners perform better than DT T-Learners.

**Learnings:** Black-box HTE models are not always the most accurate on all datasets, and interpretable segmentation methods such as VT, Distill-HTE, and R2P can be as or more accurate on some datasets. We applied this lesson at the deployment phase of our system where black-box methods and interpretable methods are considered side-by-side.

### 5.2 Comparing Policy Learning Methods

We compare the methods described in Section 4.3 to (1) a black-box policy: training a T-Learner HTE model, then assigning each individual to the treatment group with best predicted treatment effects; (2) a random policy: choosing the treatment for each unit uniformly at random. (3) On small datasets, we also compared to the PolicyTree [57] method that uses exact tree search.

**Setup:** Besides the Synthetic A dataset described in 5.1, we use other publicly-available datasets. The IHDP dataset [20], commonly used in the causal inference literature (e.g [26, 46] and many others), studied the impact of specialized home visits on infant cognition using mother and child features. A multiple-outcome dataset, Email Marketing [21] has 64k points, and visits, conversions, and money spent outcomes (see Appendix for details on how we constructed potential outcomes on this real dataset). We combine multiple outcomes as explained in Section 4.2. The ensemble policies (GUIDE-ExploreExploit, GUIDE-OPE) are based on GreedyTreeSearch-HTE and Distill-Policy – selected because of their individual performance. For the optimal tree search method, we used the implementation provided by the authors.

**Evaluation:** Since these datasets have synthetically injected potential outcomes \(Y_i^{(k)}\), for each individual we know the optimal treatment group. We then evaluate different policies \(\Pi\) using regret (against the optimal treatment): 

\[
R(\Pi) = \sum_{i=1}^{n} \max_{k} Y_i^{(k)} - Y_i^{(\Pi(X_i))},
\]

where \(\Pi(X_i)\) is the treatment group \(\in \{0, \ldots, K\}\) prescribed by policy \(\Pi\) for individual \(i\).

**Hypothesis:** The best policy generation methods should have low regret. Additionally, the more simple (less tree depth, and hence less segments) the policy, the easier it is to deploy, hence we study regret at a fixed level of complexity.

**Results:** Table 2 presents the results.

Is personalization needed? Interestingly, the majority of individuals in IHDP have positive treatment effects. Hence, there is limited benefit from personalization, where assigning all points to treatment 1 already yields the lowest relative regret of 0.36. Methods

---

1. https://github.com/vanderschaalab/noforeathlbhlpub/tree/main/alg/r2p-hte
2. https://cran.r-project.org/web/packages/policytree/index.html
with a reject option (No-HTE-Greedy, No-HTE-Iterative) also performed well as they were able to pick up on this best policy and assign all points to a single treatment. In contrast, Synthetic A exhibits some heterogeneity, with non-personalized policies (assigning all to treatment 0 or 1) having very large regret compared to BlackBox. In-terpretable policies like GUIDE that resembled GreedyTreeSearch-HTE and Distill-Policy achieved the lowest regret at 0.01, with the constituent policies themselves being not far off as well at 0.03 and 0.05 respectively. On this particular dataset, interpretable policies outperformed BlackBox policy. On Email Marketing, a real dataset, with more complicated heterogeneity, BlackBox performed the best.

**Exact search.** We ran PolicyTree [57] on IHDP and Synthetic A, two small datasets where the method did not face scalability issues. Despite performing exact search, the method does not necessarily have the smallest regret, as it is not an oracle and still has to estimate potential outcomes (only after which exact search happens).

**Learnings:** The performance of policy learning methods is dependent on the amount and type of heterogeneity in the dataset; no one policy whether interpretable, black-box, or non-personalized, consistently performed well. Since we do not a priori know how heterogeneous real datasets are, this impacted our system design, and we designed our policy generation stage to be highly modular (able to quickly add or remove different policy generation methods) as well as setup offline and online evaluation pipelines to rapidly test many different policies, thus allowing our system to adapt to the complexities of industry applications.

### 5.3 Bridging Explanations and Policies

**HTE model understanding:** Fig. 3 presents segments found by Distill-HTE on the COVID data. The segment with the most negative predicted treatment effect (first segment in red), at -0.03 ± 0.001, covers individuals who started taking the drug between 4.5 and 10.5 days after the onset of symptoms and had Aspartate Aminotransferase and Lactate Dehydrogenase levels within normal ranges [38], suggesting that they were not extremely sick. It is unsurprising that they were not predicted to benefit as much from the drug.

On the other hand, the segment with the most positive predicted treatment effect (last segment; green color) covers individuals who started taking the drug soon (≤ 4.5 days) after exhibiting COVID symptoms. These individuals were predicted to benefit the most from the drug, with treatment effect 0.018 ± 0.0006. This agrees with the finding that the Remdesivir drug results in a faster time to clinical improvement for the patients with shorter time from symptom onset to starting trial [54].

**Policy generation:** A simple policy can be derived from the Distill-HTE segments, with all red segments assigned to not receiving the drug (as they are predicted to not benefit from it) and all green segments assigned to receiving the drug. This policy is different from the interpretable policies learned directly (Table 2). For example, the GreedyTreeSearch-HTE policy is extremely simple: assigning to receive the drug only if days from symptoms onset to starting trial < 11. Another example is No-HTE-Greedy and No-HTE-Iterative whose rather stringent splitting criteria did not find enough heterogeneity worth assigning different treatment groups to, and assigned all individuals to receive the drug.

### 6 Deployment at Meta

#### 6.1 Personalizing UX Layout Using Interpretable Policies

We tested the performance of the interpretable policy generation system on a UX design problem at Meta. With these new UX designs, individuals can see more media content per page. The hope is that this change will lead to more engagement. There are two...
UX designs (i.e. treatments) under consideration. There is one goal metric (larger is better) and one guardrail metric (must not decrease). The initial A/B test showed that Treatment 1 increases the goal metric on certain cohorts, and kept the guardrail metric neutral; Treatment 2 significantly increases the goal metric, but prohibitively decreases the guardrail metric. Our goal is to find a better tradeoff that preserves the goal metric gains seen in Treatment 2, but with the neutral guardrail metrics under Treatment 1.

To generate the interpretable policies, we trained black-box HTE models (X-learners) for each treatment, and applied our policy generation algorithms on them. These policies show that individuals who seldom view or engage with this type of media content should receive Treatment 1, while individuals with a history of consuming this content should receive Treatment 2. Based on the OPE results (see Table 3 for details), two interpretable policies were selected for online tests. With these particular metrics, even small improvements were of practical significance, and one interpretable policy was launched to the product. The launch realized more than 80% of the organization’s goal on the goal metric without adversely impacting the guardrail metric.

Table 3: Offline policy evaluation (OPE) results on the UX design use case. The numbers are the estimated mean goal/guardrail metric values, larger is better. Non-personalized policies were colored in red, the two candidate personalized policies generated from our methods are in blue. The policy that delivers the best goal metric improvement is bolded. Results are normalized w.r.t. the OPE of “Assign all to control” method.

| Method                  | Test-Set OPE: Goal Metric | Test-Set OPE: Guardrail Metric |
|-------------------------|----------------------------|-------------------------------|
| Assign all to control   | 100.0 ± 0.7               | 100.0 ± 0.6                  |
| Assign all to treatment 1 | 99.2 ± 0.7               | 100.0 ± 0.6                  |
| Assign all to treatment 2 | 100.8 ± 0.7              | 99.8 ± 0.6                   |
| Distill-Policy          | 100.9 ± 0.7               | 99.9 ± 0.6                   |
| No-HTE (Greedy)         | 100.8 ± 0.7               | 100.0 ± 0.6                  |

6.2 Understanding Black-Box Personalization of Login Experience

We used our HTE model understanding method on an account login flow problem at Meta. When a person is not automatically logged into their account after clicking on a notification, they could be directed to one of three experiences: (1) a traditional login screen, (2) a one-click login flow where a code is sent via email or text message to authenticate the user, or (3) a secondary login screen to allow the user to select between one-click login or a secondary identification flow. The product team wanted to increase login success while keeping guardrail metrics neutral.

The product team employed black-box HTE modeling to personalize this login experience. When deciding to ship the black-box personalized policy, the team used our Distill-HTE method to review heterogeneity present in the user base. The analysis revealed individuals with a recent password failure, who are using a device they don’t own, or who haven’t logged in recently respond most positively to the secondary login screen, and that the secondary login screen is the best treatment for individuals sharing a device. Users who are primarily active on the web browser site prefer the traditional login screen, while active app users respond best to the one-click login treatment. Furthermore, the analysis surfaced an issue in the team’s experiment setup where a feature value was being incorrectly recorded. The team leveraged these findings to unblock shipping the personalized policy, fix the surfaced issue, and better understand the user base for future product changes.

6.3 Product Team Interviews

During development of this work, we conducted formal interviews with six potential internal customers from different product teams at Meta. The roles of the interviewees included: software engineer, data scientist, project manager, and marketer. With these backgrounds, interviewees exhibited a wide range of comfort interacting with and analyzing data. While data scientists were more comfortable with raw test output than other interviewees, all interviewees preferred small trees and the unrolled segments output shown in Figure 3 to raw text. Furthermore, those with less data analysis experience, such as marketers and project managers, preferred output like Figure 3 to small trees because interpreting trees was not immediately obvious to them. Overall, all potential customers believed the proposed interpretability analysis would benefit their understanding of HTE models and increase confidence in what the black-box models were doing for their product features at Meta.

6.4 Deployment

The Distill-HTE method is deployed within Meta, at scale on all experiments using black-box model personalization. Models for black-box personalized experiments are retrained recurrently, and the Distill-HTE method is reran after each retraining. We display the most recent analysis results, as shown in Figure 3, within the experimentation system UI, thus allowing for an intuitive user experience. Additionally, we have deployed interpretable policies for multiple products at Meta. To deploy these policies, we use a modified version of the framework that deploys the black-box policies. Since interpretable policies are inherently simple to maintain interpretability, we can replace the black-box policy call with a few basic if-else statements. In practice, we implement the if-else blocks with comparison operators (less than, greater than, etc.). Product teams frequently consider interpretable policies alongside black-box policies.

The choice between black-box policies derived from HTE models and interpretable policies often depends on the ability of product teams utilizing such policies to maintain personalization in production and the need to explain how exactly personalization is happening. While HTE models are resource and maintenance intensive, the ability to continuously retrain the model allows for adjustment to a dynamic user base. Conversely, interpretable policies are easy to implement and maintain, but may not perform best over the long-term without policy regeneration as the user base changes. Additionally, we have found that interpretable policies can be a better fit when app startup time is a constraint.

7 CONCLUSION

The motivation for this work was three-fold. (1) HTE models sometimes overfit on extremely large datasets, and can be hard to interpret; (2) Interpretable policies can avoid some of the tech debt that black-box policies incur [45]. (3) Scalability constraints exclude...
optimal tree-based policy learning algorithms from deployment in production environments. Our two-stage system that understands black-box HTE models and generates interpretable personalized policies is deployed at Meta and runs on all personalized experiments. We hope that the practical experience and lessons shared in this paper can help other organizations wishing to incorporate interpretability techniques into their experimentation systems.

Acknowledgements: The authors would like to thank Rishav Rajendra, Zheng Yan, Chad Zhou, Sam Howie, Norm Zhou, and Ariel Einvine for valuable discussion and collaboration.

REFERENCES

[1] Maxime Amram, Jack Dunn, and Ying Daisy Zhuo. 2020. Optimal Policy Trees. arXiv:2012.02729 (2020).
[2] Susan F Assmann, Stuart J Pocock, Laura E Enos, and Linda E Kasten. 2000. Subgroup analysis and other (mis) uses of baseline data in clinical trials. The Lancet 355, 9209 (2000).
[3] Susan Athey and Guido Imbens. 2016. Recursive partitioning for heterogeneous causal effects. PNAS 113, 27 (2016).
[4] Susan Athey and Stefan Wager. 2021. Policy learning with observational data. Econometrica 89, 1 (2021), 133–161.
[5] Dimitris Bertsimas, Jack Dunn, and Nishanth Mundru. 2019. Optimal prescriptive trees. INFORMS Journal on Optimization 1, 2 (2019), 164–183.
[6] Max Biggs, Wei Sun, and Markus Zitt. 2021. Policy Distillation for Revenue Optimization: Interpretable Personalized Pricing. In ICML.
[7] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In KDD.
[8] Rich Caruana. 1997. Multitask learning. Machine Learning 28, 1 (1997), 41–75.
[9] Gong Chen, Hua Zhong, Anton Belousov, and Viswanath Devanarayan. 2015. A PRIM approach to predictive-signature development for patient stratification. Stat Med 34, 2 (2015), 317–342.
[10] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In RecSys.
[11] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly Robust Policy Evaluation and Learning. In ICML.
[12] Raaz Dwivedi, Yan Shuo Tan, Briton Park, Mian Wei, Kevin Horgan, David Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly Robust Policy Learning: Generalization and Optimization. In ICML.
[13] Florent Garcin, Christos Dimitrakakis, and Boi Faltings. 2013. Personalized News Recommendation with Context Trees. In RecSys.
[14] Antonino Freno. 2017. Practical Lessons from Developing a Large-Scale Recommender System in the get-out-the-vote campaign. Political Analysis 19, 1 (2011), 1–19.
[15] Nathanael Jo, Sina Aghaei, Andrés Gómez, and Phebe Vayanos. 2021. Learning Optimal Prescriptive Trees from Observational Data. arXiv:2108.13628 (2021).
[16] Fredrik Johansson, Utri Shalit, and David Sontag. 2016. Learning representations for counterfactual inference. In IML.
[17] Nathan Kallus. 2017. Recursive partitioning for personalization using observational data. In ICM.
[18] Nathan Kallus and Angela Zhou. 2018. Policy evaluation and optimization with continuous treatments. In AISTATS.
[19] Edward H. Kennedy. 2020. Optimal doubly robust estimation of heterogeneous causal effects. arXiv:2004.14497 (2020).
[20] Toru Kitagawa and Alexey Teterov. 2018. Who should be treated? empirical welfare maximization methods for treatment choice. Econometrica 86, 2 (2018).
[21] Sören R Künzel, Jasjeet S Sekhon, Peter J Bickel, and Bin Yu. 2019. Metalearners for estimating heterogeneous treatment effects using machine learning. PNAS 116, 10 (2019), 4156–4165.
[22] Akos Lada, Alexander Peysekachovik, Diego Aparicio, and Michael Bailey. 2019. Observational data for heterogeneous treatment effects with applications to recommender systems. In Elsevier.
[23] Hyafil Laurent and Ronald L Rivest. 1976. Constructing optimal binary decision trees for heterogeneous treatment effect estimation. In AAI.
[24] Kevin Hillstrom. 2008. MineThatData E-Mail Analytics And Data Mining Challenge.
[25] Nyoman Gunantara. 2018. A review of multi-objective optimization: Methods and its applications. In JMLR.
[26] Susan Athey and Guido Imbens. 2016. Recursive partitioning for heterogeneous causal effects. PNAS 113, 27 (2016).
[27] Susan Athey and Stefan Wager. 2021. Policy learning with observational data. Econometrica 89, 1 (2021), 133–161.
[28] Dimitris Bertsimas, Jack Dunn, and Nishanth Mundru. 2019. Optimal prescriptive trees. INFORMS Journal on Optimization 1, 2 (2019), 164–183.
[29] Max Biggs, Wei Sun, and Markus Zitt. 2021. Policy Distillation for Revenue Optimization: Interpretable Personalized Pricing. In ICML.
[30] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In KDD.
[31] Rich Caruana. 1997. Multitask learning. Machine Learning 28, 1 (1997), 41–75.
[32] Gong Chen, Hua Zhong, Anton Belousov, and Viswanath Devanarayan. 2015. A PRIM approach to predictive-signature development for patient stratification. Stat Med 34, 2 (2015), 317–342.
[33] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In RecSys.
[34] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly Robust Policy Evaluation and Learning. In ICML.
[35] Raaz Dwivedi, Yan Shuo Tan, Briton Park, Mian Wei, Kevin Horgan, David Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly Robust Policy Learning: Generalization and Optimization. In ICML.
[36] Florent Garcin, Christos Dimitrakakis, and Boi Faltings. 2013. Personalized News Recommendation with Context Trees. In RecSys.
[37] Antonino Freno. 2017. Practical Lessons from Developing a Large-Scale Recommender System in the get-out-the-vote campaign. Political Analysis 19, 1 (2011), 1–19.
[38] Nathanael Jo, Sina Aghaei, Andrés Gómez, and Phebe Vayanos. 2021. Learning Optimal Prescriptive Trees from Observational Data. arXiv:2108.13628 (2021).
[39] Fredrik Johansson, Utri Shalit, and David Sontag. 2016. Learning representations for counterfactual inference. In IML.
[40] Nathan Kallus. 2017. Recursive partitioning for personalization using observational data. In ICM.
[41] Nathan Kallus and Angela Zhou. 2018. Policy evaluation and optimization with continuous treatments. In AISTATS.
8 APPENDIX

8.1 Algorithm Details

In this section we provide more details of algorithms described in Section 4.3. Define $\hat{M}_j(S) = \sum_{i \in S} \hat{Y}_i^{(j)}$ and $\hat{M}(S) = \max_j \hat{M}_j(S)$.

**Algorithm 1: GreedyTreeSearch-HTE**

*Input*: Maximal tree depth $d$, dataset $\mathcal{D}$, predictions from HTE model $(\hat{Y}_i^{(0)}, \ldots, \hat{Y}_i^{(K)})$ for $i \in \mathcal{N}_D$.

*Output*: A list of segments: $\text{segments}$.

1. Set depth := 0 and $\text{segments}$ := $\mathcal{D}$.
2. Add segments to $\text{segments}$ using greedy search:
   - Set nodes := $\{}$
     - for $S \in \text{segments}$ do
       - Split $S$ into $S_l$ and $S_r$ such that:
         $$(S_l, S_r) = \arg\max_{S_l, S_r} \hat{M}(S_l, i) + \hat{M}(S_r, j > i).$$
       - If such $S_l$ and $S_r$ exist, we add them to nodes;
       - otherwise we add $S$ to nodes.
   - Let segments := nodes and depth := depth + 1.

---

**Algorithm 2: No-HTE-Greedy**

*Input*: Maximal tree depth $d$, dataset $\mathcal{D}$.

*Output*: segments.

1. Set depth := 0 and $\text{segments}$ := $\mathcal{D}$.
2. Split the segments:
   - while depth $\leq d$ do
     - Set nodes := $\{}$
       - for $S \in \text{segments}$ do
         - Consider splits that satisfy $\min(A(S_l), A(S_r)) > A(S)$, find the optimal split defined as follows:
           $$(S_l, S_r) = \arg\max_{S_l, S_r} \min(A(S_l), A(S_r)).$$
         - If such $S_l$ and $S_r$ exist, we add them to nodes;
         - otherwise add $S$ to nodes.
     - Let segments := nodes and depth := depth + 1.

---

**Algorithm 3: No-HTE-Iterative**

*Input*: Maximal tree depth $d$, number of iterations $t$, dataset $\mathcal{D}$.

*Output*: segments.

1. Set iterations := 0 and segments := $\{}$.
2. Add to segments iteratively
   - while iterations $\leq t$ do
     - Step 1 to 2 in Algorithm 2, but change all
     - Denote the resulting list of segments as $\mathcal{C}$.
     - Add to segments the $S$ in $\mathcal{C}$ that maximizes $A(S)$ and remove the data points in $S$ from $\mathcal{D}$.
   - Set iterations := iterations + 1.

---

**Algorithm 4: GUIDE-ExploreExploit**

*Input*: Maximal tree depth $d$, dataset $\mathcal{D}$, predictions $(\hat{Y}_i^{(0)}, \ldots, \hat{Y}_i^{(K)})$ for $i \in \mathcal{N}_D$, $Q$ trained interpretable policies $\Pi_1, \ldots, \Pi_Q$.

*Output*: A list of segments: $\text{segments}$.

1. Generate a dataset with randomly selected policies on individuals
   - for $i = 1$ to $N$ do
     - Randomly select a policy from $\Pi_1, \ldots, \Pi_Q$. Let $A_i$ be the index of this policy, $A_i \in \{1, \ldots, Q\}$. Apply this policy to individual $i$, assume the policy assigns treatment group $k \in \{0, 1, \ldots, K\}$. Initialize $\Omega_i = [0, \ldots, 0] \in \mathbb{R}^{K+1}$. Let $W_i$ be the treatment received by individual $i$ in the dataset.
     - If $k = W_i$, set $O_i^{(k)} := Y_i$. Otherwise, set $O_i^{(k)} := \hat{Y}_i^{(k)}$.
   - We now have a new dataset of the form $(X_i, A_i \in \{1, 2, \ldots, Q\}, \Omega_i)$.
2. Apply Algorithm 1 to the new dataset, using
   - $(O_i^{(0)}, \ldots, O_i^{(K)})$ in place of $(\hat{Y}_i^{(0)}, \ldots, \hat{Y}_i^{(K)})$.

---

8.2 Ensemble Algorithm Details

In this section we provide more details of algorithms described in Section 4.3.4. Suppose we have access to $Q$ policies $\Pi_1, \Pi_2, \ldots, \Pi_Q$. We wish to train an ensemble policy $\hat{\Pi}$ that uses all or a subset of the policies $\Pi_1, \Pi_2, \ldots, \Pi_Q$ while still remaining interpretable. For ease of notation, we assume the trained policies $\Pi_1, \ldots, \Pi_Q$ were obtained from another split of the dataset and we can safely use $\mathcal{D}$ as the validation set on which we learn the ensemble policy.

**GUIDE-ExploreExploit** is inspired by the explore-exploit paradigm in the contextual bandits literature [47]. To perform this offline, we use HTE outcome predictions when the observed outcome is not available in the dataset. See Algorithm 4 for the implementation. The ensemble policy $\hat{\Pi}$ is generated using Algorithm 1 but with the HTE predictions $(\hat{Y}_i^{(0)}, \ldots, \hat{Y}_i^{(K)})$ replaced by $(O_i^{(0)}, \ldots, O_i^{(K)})$.

**GUIDE-OPE** generates a policy ensemble by maximizing off-policy evaluation (Equation (2)) at each split. Here, we aim to find one feature split such that the left and right children uses a different candidate policy. Concretely, suppose we split dataset $\mathcal{D}$ into $\mathcal{D}_L$ and $\mathcal{D}_R$, let $\hat{\Pi}$ be the ensemble policy that applies $\Pi_k$ to $\mathcal{D}_L$ and $\Pi_q$ to $\mathcal{D}_R$. To create $\hat{\Pi}$, we use exhaustive search to find the optimal feature split and candidate policies $(\hat{\Pi}_1^*, \hat{\Pi}_2^*, \hat{\Pi}_3^*, \hat{\Pi}_q^*)$ that solves max $D(\hat{\Pi} \mathcal{D}_L, \mathcal{D}_R, \hat{\Pi}_1^*, \hat{\Pi}_2^*, \hat{\Pi}_3^*, \hat{\Pi}_q^*)$.
8.3 Data Generation Details

The Email Marketing dataset [21], first introduced in the public MineThatData Data Mining challenge, is a real dataset from an experiment where customers were randomized into receiving one of three treatments. To generate potential outcomes, for each individual $i$ in treatment group $k$, we searched for the 5-nearest-neighbors in treatment group $k'$, averaging their outcomes to get the potential outcome for individual $i$ for treatment group $k'$. To compute distance between individuals, we used Euclidean distance in feature space.

8.4 Training Details

Unless otherwise mentioned, the HTE models we train are T-learners consisting of GBDT base learners. In general, we use 40% of the data as the test set on which we report results, and 30% of the remaining 60% as the validation set. We learn individual policies on the training set. When learning ensemble policies, we learn the ensemble on the validation set.

1https://blog.minethatdata.com/2008/03/minethatdata-e-mail-analytics-and-data.html