Exhibit 19

To alleviate the computational cost, we further decompose ranking into multiple stages, so the best (and high cost) features and models can focus only on a few top candidates. For example, an **early-stage** ranking model is lightweight and handles 1K~5K candidates, followed by a more complex **late-stage** ranking model that handles ~100 candidates. This setup gives us flexibility, but involves trade-offs on accuracy.

Value Model / Slate Composition. Individual model predictions like `p(like)`, `p(share)` from the ranking models need to be combined with other factors to produce a final, ordered "slate" of content to show to the user. We call this mechanism the "Value Model".

The Value Model considers the prediction scores, quality scores, along with other factors (e.g., integrity, diversity of the results, creators covered, fairness, etc.) to give the final result. It can also do things like enforcing certain rules, boosting certain types of content, etc. This mechanism is a combination of models and human designs; the most complex Value Model used at Meta is in Feed, with more than 2,000 lines of code. Typically, people across functions collaborate on it, to solve challenges like balancing different objectives and metrics, aligning them towards the overall product "value" which may not be mathematically clear yet, and making changes to fix urgent problems.

We can enhance the Value Model to consider more personalized preferences. We could even learn models to directly compose the slate to optimize the desired objectives automatically. This approach is less used at Meta right now due to significant research and infrastructure challenges, but there are active explorations.

The final slate is sent to the app for the user to view, and we collect their feedback to train the models better. It is possible to further adjust the slate in the App, either using heuristics (e.g. skip heavy videos when the network is slow), or possibly using on-device ML (e.g. Federated Learning) in the future.

The discussion so far is more focused on short-term engagements, and assumes a supervised learning setting where the prediction targets are well-defined, immediately observable outcomes. It is very useful to also optimize for other, more long term oriented objectives, e.g., user satisfaction as measured by various surveys, and retention as measured by DAP. We discuss these more in the deep dive and future sessions.

Interlude: How the Models are Made

So far we talked about how the models were making all kinds of predictions. Here we describe a simple case to illustrate how these models are made. This process corresponds to the right side of Fig 2.

First, we construct **Training Data**, which is the source of knowledge for the models to learn. The majority of the training data are based on user **feedback** from the App, like how a user engaged with a piece of content. Training data are usually organized by a table, where each row (called an "example") represents the observations around an outcome. Each example has two parts: 1) **Label**, the key outcome and what we want to predict, 2) Features, descriptions of the scenario where the outcome happened. Below is a toy training data table for predicting whether the user would like a movie recommendation. In reality, our training tables consist of tens of billions of rows and hundreds to thousands of features, and take terabytes of space to store.

Label	Features	
Liked = yes	UserID=12345, Age, Time, Watch_History=[StarWars, Mission Impossible, Matrix, …], Following=[SiFiFans, ActionFever, …], …	VideoID=Batman, ProducerID, Genre=SciFi, Director, Popularity=High, …

Liked = no	UserID=12345, Age, Time, Watch_History=[StarWars, Mission Impossible, Matrix, Batman…], Following=[SiFiFans, ActionFever, …], …	VideoID=AmericanBeauty, ProducerID, Genre=Drama, Director, Popularity=Mid, …

The next step is **Model Training**, where the model ingests and learns the training data, so that it can predict the label based on the provided features. Note that in the training data, both label and features are there, since it's in the past and everything is observable. But during serving, the model needs to predict the label outcome that would happen in the future, based only on the features. On the high level, training is much like an error correction process: for each example, the model would make a prediction, and compare it to the true label. If it's wrong, the model is adjusted so the prediction gets closer to the label.

Training is a complex and expensive step, as the models need to learn about billions of users, millions / billions of items, from tens of billions of past examples. Dedicated training algorithms, software, and even hardware, are designed to do this, so that we can support many different model architectures, train fast, and the model can 1) converge (reach the optimum) fast, 2) avoid overfitting (the model fits the past data, but cannot predict the future well), 3) learn new information without forgetting about the past.

After training, the model is sent to **Serving** to make predictions (dubbed "**Model Publishing**"). This is also a complex process to make the models that were trained on one software/hardware stack, which can be terabytes in size with complex structures, to run on a different stack in a scalable and efficient way. Steps include: post-processing (e.g. quantization), decomposition, storage & transportation, and then distribution and reconfiguration on the fleet of serving machines. This can also be done in an online fashion, where live data is streamed into the model training and the model is continuously updated and synchronized to serving. Using online learning we can learn fresh information more quickly, without relearning the old data. One important factor here is the training loop latency, which means the delay between observing a user feedback, and using its learnings to make (better) predictions. Low latency is important when we want to quickly respond to the users' changing preferences, or adjust delivery for new content based on fresh data.

Model **Architecture Designs** (e.g. here) are done by researchers and engineers to capture the relationships among the labels and features to best predict future labels. We have very active research efforts here to produce innovative and impactful designs. Model Design, Training, Data, and Feature need to work in tandem, and the final prediction quality is determined by the weakest link; a naive model will let the rich information in the data go to waste, on the other hand, even the best models can't make up for bad data. The development process is usually iterative and interweaves multiple factors, and we want to move fast, but that often poses challenges to our software and hardware stacks to achieve the desired scalability, flexibility, and reliability.

Producer Journey

We want to make producers eager to participate and feel that they have an opportunity to thrive on our platform. We inspire producers to create via showing them mimicable content. We give their content exposure, and ensure they receive feedback by matching them to the right audience. We leverage consumer feedback to identify the best content and help it shine on our platform. Good producer experiences help retain and grow quality producers, thereby enriching our inventory and benefitting the overall ecosystem.

Below is the high level process of distributing content for producers. It's generally embedded in the previous system, but with special designs for producers, such as producer-oriented generators in Retrieval, producer outcome predictions in Ranking, faster training loops for new producers and content, and Value Model designs to ensure they get sufficient exposure.