Token-Based Conjoint –
A New Framework for Too Many Attributes

Megan Peitz – Numerious Inc.
Trevor Olsen – Numerious Inc.

Abstract

This paper introduces Token-Based Conjoint (TBC), a novel approach for conducting conjoint analysis when the number of product features is exceptionally large (30+ attributes). TBC draws inspiration from MaxDiff and choice-based conjoint (CBC), asking respondents to evaluate small subsets of features by allocating tokens to their top preferences, followed by a purchase likelihood question. This structure enables respondents to meaningfully engage with complex feature sets while generating data amenable to additive modeling.

We validated TBC across three studies involving up to 40 features, assessing its predictive power through both in-sample and out-of-sample tasks. Results demonstrated strong accuracy and offered clear insights into feature-level drivers of uptake. We also explored ways to scale TBC further, testing simplified task structures and a weighted modeling approach to mitigate overfitting when task exposure varies.

TBC is not a one-size-fits-all method. In scenarios where price is a central focus, such as revenue optimization or willingness to pay per feature, other methods may be more suitable.

Overall, TBC provides a practical, respondent-friendly solution for high-dimensional feature prioritization. It is especially well-suited for uncovering additive value when pricing isn't the primary objective — particularly in settings where diminishing returns and utility saturation are key considerations.

Introduction

One of the most common questions in product research is: *"Which combination of benefits will make people buy my product?"* This was exactly the challenge posed by a client who wanted to identify the optimal feature bundle for a new offering. Conjoint analysis was the obvious starting point as it's designed to reveal how combinations of features drive preference. But when we asked how many benefits they wanted to test, the answer came back: *"Maybe 30, even up to 40."*

At that scale, traditional conjoint methods start to break down. Presenting respondents with dozens of attributes leads to cognitive overload and unreliable data. Even advanced designs that try to accommodate large attribute sets have issues: they often produce profiles with unrealistic balance (e.g., every product having the same number of features "on") and fail to capture diminishing returns when stacking multiple high-utility features.

MaxDiff, or best-worst scaling, is another common alternative. It handles long feature lists well and can rank features by importance. But MaxDiff results are not additive, and do not model purchase likelihood. It doesn't tell us how feature *combinations* perform. We might learn that Feature A is more appealing than Feature B, but we can't predict how much more attractive a product becomes if it includes *both*. Without additive structure, we can't simulate bundle-level uptake.

This gap between conjoint and MaxDiff led us to develop Token-Based Conjoint (TBC) — a new framework that blends the strengths of both methods. In a TBC, respondents evaluate small subsets of features, "spend tokens" on their favorites, and then indicate whether that set would be enough to trigger a purchase. Repeating this across multiple feature subsets allows us to gather additive data on feature value without overwhelming respondents.

In this paper, we introduce the TBC approach and share findings from three large-scale studies involving up to 40 features. We begin by detailing how TBC works, including task design, implementation, and how data is coded for modeling. We then evaluate its performance using fixed holdout tasks and out-of-sample prediction, demonstrating strong predictive accuracy. Next, we explore how TBC can be scaled for even larger or more complex scenarios by modifying the task structure and applying a likelihood weighting strategy to maintain balance. We conclude with key takeaways, limitations, and practical guidance for applying TBC in future product research.

Methodology: The Token-Based Conjoint Approach

Design of the TBC Task

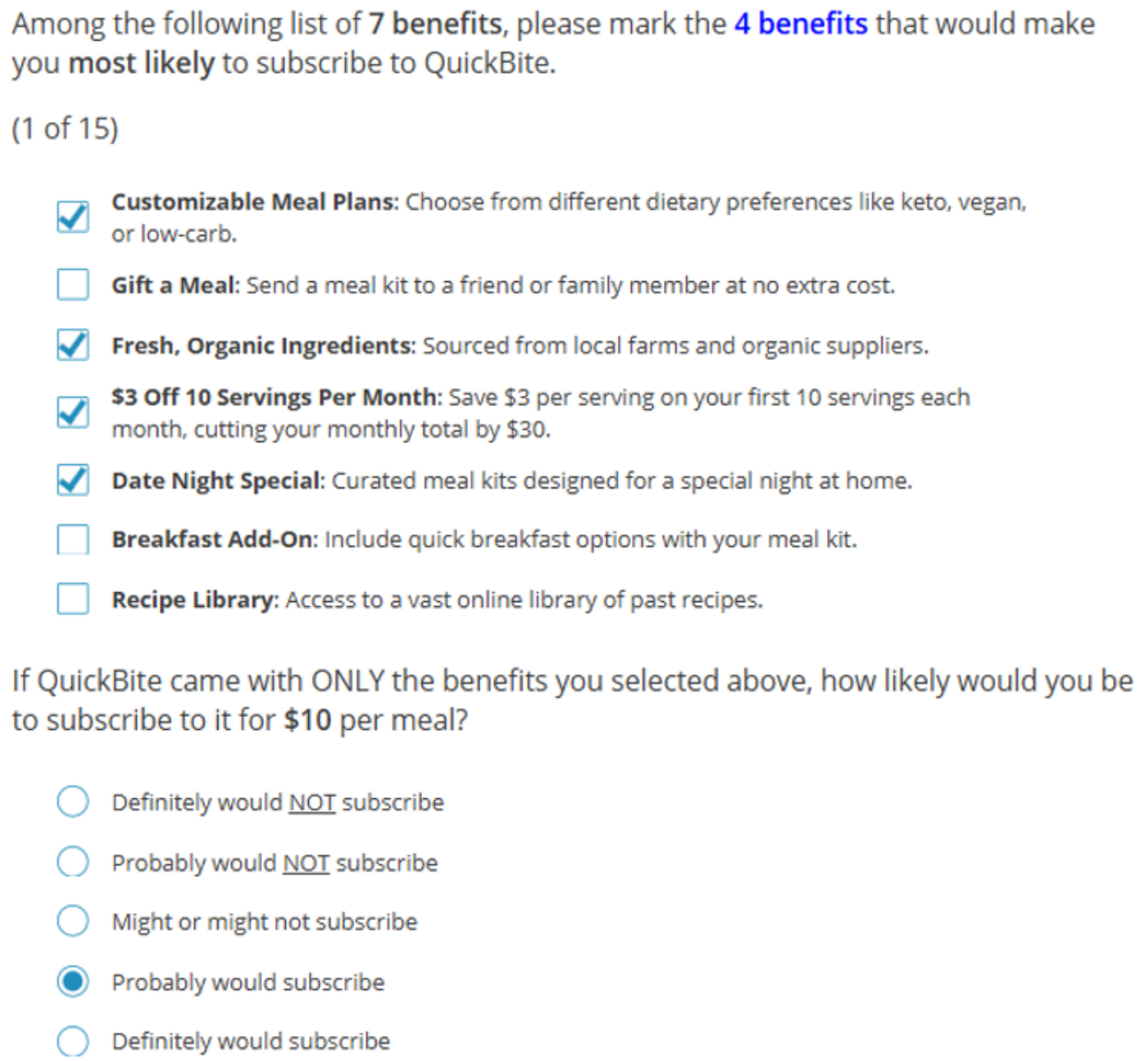
In a Token-Based Conjoint (TBC) exercise, respondents evaluate manageable subsets of product features and select their top preferences. Each screen presents n features, and respondents are asked to choose the k features they value most. A key innovation is that k is dynamic and varies from task to task. One screen might ask the respondent to pick just one feature ($k = 1$), the next might ask for two ($k = 2$), and so on, up to $k = n-1$.

This "allocate k tokens out of n" design mirrors real-world decision-making: consumers rarely get everything they want and must make trade-offs. By varying k across tasks, we allow

respondents to signal both their absolute top choices and the broader set of features they find valuable under different constraints.

After each selection, we include a 5-point purchase likelihood question. This follow-up question is a simplified, purchase-based framing of the "None" option, similar to dual-response none formats used in conjoint. It captures whether a respondent's selected bundle is strong enough to trigger a real purchase decision. We used a five-point scale ranging from "Definitely would subscribe" to "Definitely would not subscribe," providing a nuanced measure of intent.

Image 1 - Example TBC Screen



Why This Design?

The TBC task is designed to balance cognitive simplicity with realistic trade-offs. By limiting each screen to ***n*** features, we avoid overwhelming respondents with the full list of attributes. No matter how long the master list is, the respondent only evaluates a manageable subset at a time.

The use of a dynamic ***k***, where respondents select different numbers of features on each screen, adds further depth. It allows us to observe preferences for both small, focused bundles and larger, more inclusive ones, all within the same exercise. This variation also prevents over-selection bias: even highly desirable features must compete to be the *one and only* pick when ***k*** = *1*, but may be chosen alongside others when ***k*** = *5 or 6*. This helps uncover diminishing returns, where the value of a feature might be high in isolation but less impactful when bundled with similar benefits.

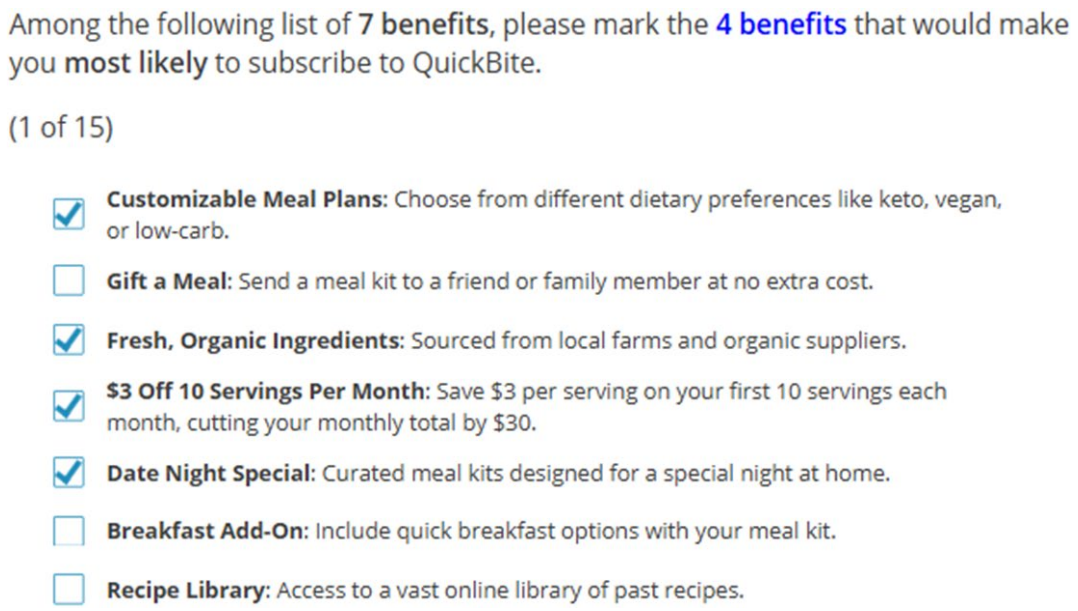
The purchase likelihood follow-up question grounds the task in a real-world decision: *Would this specific bundle be good enough to buy?* Unlike traditional best-worst or rank-based tasks, this step introduces an absolute threshold. Some bundles pass the bar; others don't. That binary framing — *buy or not buy* — feeds additive purchase information into the model, much like the "None" option in a traditional conjoint, allowing us to simulate both relative feature appeal and overall purchase likelihood.

Implementation

We implemented the TBC exercise using Sawtooth Software's Lighthouse Studio. Technically, we configured the task as a looped free-format question based on a MaxDiff design with a constructed list to manage the dynamic number of selections.

A balanced MaxDiff design was generated with the total number of features, ensuring that each feature appeared across screens a consistent number of times and in varied combinations with others. Instead of asking respondents to select a "best" and "worst" item (as in standard MaxDiff), we modified the prompt to say:

Image 2 - Token Exercise



We used Lighthouse's constructed list functionality to pipe in the correct ***k*** value for each screen dynamically.

Once a respondent selected their top ***k*** features, they were immediately shown a follow-up question (See Image 3).

Image 3 - Purchase Likelihood Follow-Up Question

This follow-up links the feature selection directly to an outcome, injecting purchase intent into the dataset alongside preference information.

Coding the Data for Analysis

TBC responses can be analyzed using a standard discrete choice modeling framework by treating each screen as a constructed choice task among possible feature bundles. When a respondent sees ***n*** features and selects ***k*** of them, we interpret their chosen set as a single "alternative" in a choice task.

To define the full choice set, we enumerate all possible combinations of size ***k*** that could have been selected from the ***n*** features shown. For example, if ***n*** = *5* and ***k*** = *2*, there are "5 choose 2"

$$\binom{5}{2} = 10$$

possible 2-feature combinations. A respondent chose one of these combinations and implicitly rejected the other nine.

For analysis, we can construct a choice task with these 10 alternatives: assign a 1 to the selected bundle and 0 to the others. This allows us to estimate the model using approaches like multinomial logit (MNL) or hierarchical Bayes MNL, treating each bundle as an observed choice from a well-defined set of alternatives.

Image 4 - Coding of a 5 choose 2 task where respondent picked item 2 and 4

Task	Alternative	Item #					Winner	
		1	2	3	4	5		
1	1	0	**1**	0	**1**	0	1	← Winning Pair
1	2	1	**1**	0	0	0	0	
1	3	0	**1**	1	0	0	0	
1	4	0	**1**	0	0	1	0	
1	5	1	0	0	**1**	0	0	
1	6	0	0	1	**1**	0	0	
1	7	0	0	0	**1**	1	0	
1	8	1	0	1	0	0	0	No winning items
1	9	1	0	0	0	1	0	
1	10	0	0	1	0	1	0	

Incorporating the Purchase Likelihood ("None") Response

To capture not just what features respondents prefer, but whether their chosen bundle is actually good enough to purchase, we include a follow-up purchase likelihood question after each selection. This allows us to incorporate a "None" option into the modeling framework, indicating whether the respondent would buy the bundle or not.

The standard approach, often referred to as dual-response none coding (Diener et al., 2006), handles this as follows:

- If the respondent would not purchase the bundle, we model two tasks:

    1. A choice among the possible feature combinations (with the chosen bundle coded as selected), and

    2. A second task including "None" as an alternative, coded to show the respondent explicitly chose "None."

- If the respondent would purchase the bundle, we model just one task: a choice that includes the "None" option, with the selected bundle preferred over it.

This structure helps the model estimate both relative preferences among features and the absolute threshold required to drive a purchase.

In our implementation, we tested an alternative approach called the naïve none. Here, we generate two modeled tasks for every screen, regardless of the respondent's purchase likelihood:

1. A standard choice among feature bundles (as described above), and

2. A binary task comparing the chosen bundle directly against "None."
   If the respondent said they would purchase, the bundle is marked as preferred; if not, "None" is.

Unlike the standard method, the naïve none consistently creates two analytic choice sets for all screens. This uniformity ensures that each respondent contributes the same amount of information—one feature selection task and one purchase decision task per screen.

Image 5 – Adding the None Coding

	Task	Alternative	Item #					None	Winner
			1	2	3	4	5		
Respondent says they would buy	2	1	0	0	0	0		1	0
	2	2	0	1	0	1	0	0	1
Respondent says they would NOT buy	2	1	0	0	0	0		1	1
	2	2	0	1	0	1	0	0	0

Handling the 5-Point Purchase Likelihood Scale

To capture nuanced purchase intent, we asked respondents to rate how likely they would be to subscribe or purchase using a five-point scale, ranging from *"Definitely would subscribe"* to "*Definitely would not subscribe*." This opened the door to different ways of translating those responses into binary "buy vs. not buy" decisions in the model.

We explored three coding strategies:

- Top-Box None: Only the most enthusiastic response—*"Definitely would subscribe"*—was treated as a purchase. All other responses were coded as a "no purchase," meaning "None" is preferred in the model.

- Top-2-Box None: This more inclusive threshold counted both *"Definitely"* and *"Probably would subscribe"* as indicating a purchase. Any weaker response was still treated as a "no purchase."

- Calibrated Likelihood: Instead of a binary cutoff, this approach assigns partial credit to purchase likelihood. For example, *"Definitely"* might correspond to a 70% likelihood of buying, and *"Probably"* to 30%. In this case, the model distributes the outcome weight proportionally between the selected bundle and the "None" alternative.

Image 6 - Different None Coding Strategies

The choice you made in the DR	Top Box	Top 2 Box	Calibrated
1 – Definitely would not subscribe	0	0	0
2 – Probably would not subscribe	0	0	0
3 – Might or might not subscribe	0	0	0
4 – Probably would subscribe	0	1	.3
5 – Definitely would subscribe	1	1	.7

Image 6A – Coding the Model with a Top Box Choice

	Task	Alternative	Item # 1	2	3	4	5	None	Winner	
Respondent picks top box in the top box model	2	1	0	0	0	0		1	0	*Would NOT be included if Diener Coding*
	2	2	0	1	0	1	0	0	1	
Respondent did not pick top box in top box model	2	1	0	0	0	0		1	1	
	2	2	0	1	0	1	0	0	0	

Image 6B – Coding the Model with a Top 2 Box Choice

	Task	Alternative	Item # 1	2	3	4	5	None	Winner	
Respondent picks top 2 box in the top 2 box model	2	1	0	0	0	0		1	0	*Would NOT be included if Diener Coding*
	2	2	0	1	0	1	0	0	1	
Respondent did not pick top 2 box in top 2 box model	2	1	0	0	0	0		1	1	
	2	2	0	1	0	1	0	0	0	

Image 6C – Coding the Model with a 70/30 Calibration

	Task	Alternative	Item #					None	Winner
			1	2	3	4	5		
Respondent picks top box in the calibrated model	2	1	0	0	0	0		1	.3
	2	2	0	1	0	1	0	0	.7
Respondent picks 2nd box in the calibrated model	2	1	0	0	0	0		1	.7
	2	2	0	1	0	1	0	0	.3
Respondent did not pick top 2 box in calibrated model	2	1	0	0	0	0		1	1
	2	2	0	1	0	1	0	0	0

These none-response coding schemes were part of a secondary exploration, aimed at understanding how the model responds to different purchase thresholds. While performance varied slightly across approaches, it's important to note that accuracy alone doesn't reveal which threshold is "correct." Instead, these options provide flexibility for researchers to align their modeling assumptions with prior knowledge or external calibration targets.

Baseline TBC Configuration

Throughout this paper, we refer to the “baseline” TBC model as the original version used across all three studies. It includes:

- The full ***n***-choose-***k*** task structure (i.e., the complete set of possible bundles on each screen)
- Naïve none coding (a binary choice between the selected bundle and “None” after every task)

Study Design and Data

Real-World Data

We evaluated the TBC method across three real-world client projects, anonymized and reframed for this paper as if they were conducted for a fictional meal subscription brand called “QuickBite.” While all feature descriptions were generalized, the core design structure and data characteristics were preserved.

Each project involved a long list of potential product benefits:

- Projects A and B tested 38 features each
- Project C tested 40 features

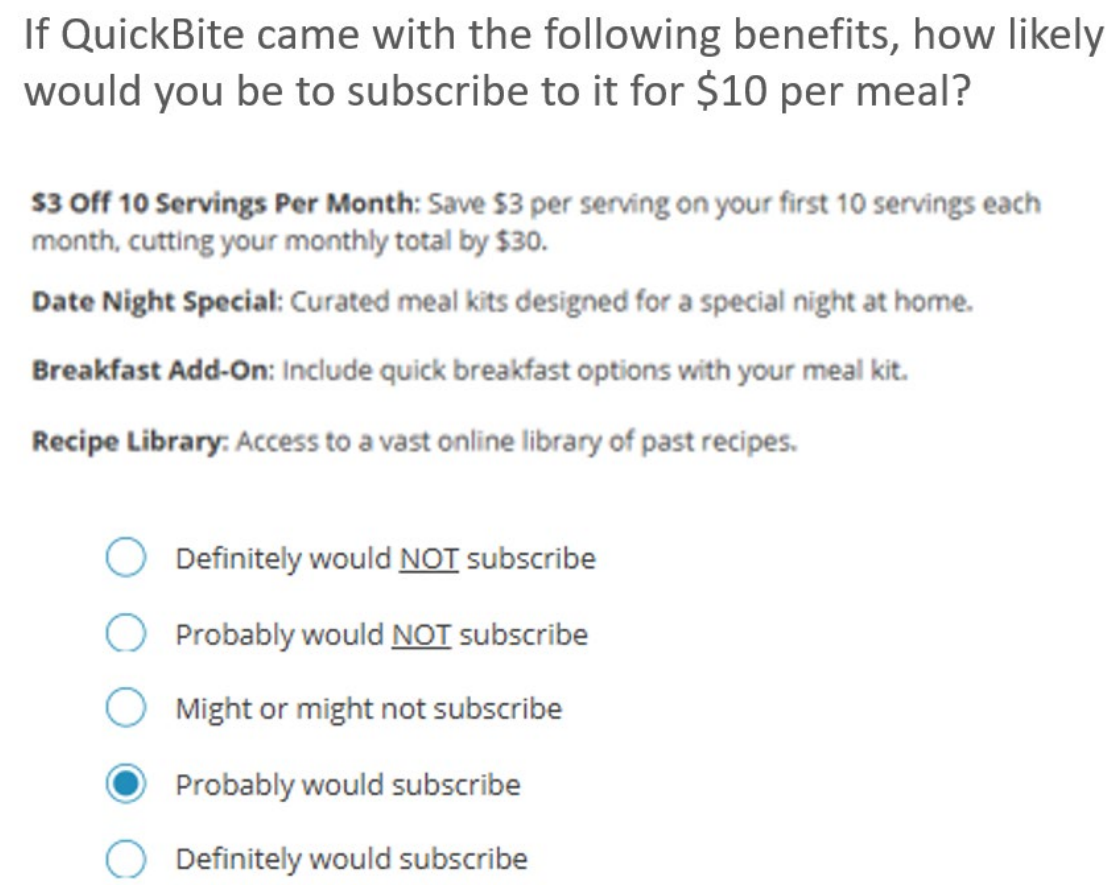
In every case, respondents completed a TBC exercise where $\boldsymbol{n}$ = 7 features were shown per screen, and they were asked to select a dynamic number of top features ($\boldsymbol{k}$)—ranging from 1 to 6. For example, one screen might ask for their single top pick ($\boldsymbol{k} = 1$), while another might ask for their top five ($\boldsymbol{k} = 5$).

- Projects A and B included 15 selection tasks per respondent
- Project C included 12 tasks
- Sample sizes:
    - Project A: ~500 (UK sample)
      Project B: ~500 (US sample)
      Project C: 812 (US sample)

Holdout Tasks for Validation

To benchmark predictive accuracy, we included two fixed holdout tasks in each survey. These tasks were the same for all respondents and presented a realistic bundle of features along with a purchase intent question, such as:

Image 7 - Sample Holdout Screen



Respondents answered using the same 5-point scale used elsewhere in the survey. Because everyone saw the same holdouts, we can directly compare actual responses to the model's predicted purchase likelihood, making these tasks a useful anchor for validation.

Validation Strategy

We used two key validation methods to evaluate model performance:

1. Holdout Task Prediction (In-Sample and Out-of-Sample)

   Using each respondent's estimated part-worth utilities, we predicted their probability of subscribing to each holdout bundle. We then compared these predictions to actual responses, calculating mean absolute error (MAE) as the metric.

   - *In-sample*: Model estimated on the full dataset

- *Out-of-sample*: 4-fold cross-validation, where each respondent is predicted using a model trained on a different 75% of the sample

2. Likelihood-Based Validation (Preference Ranking)

   To assess how well the model captures relative preference ordering, we ran a leave-one-task-out test. For each screen number, we held it out for all respondents from estimation. We then used the model to compute the log-likelihood of the actual choice in that task. Repeating this across all tasks and averaging the results gave us a measure of how well the model ranks bundles—even for unseen combinations.

Together, these two metrics — MAE and log-likelihood — help evaluate both calibration (predicting how likely someone is to buy) and discrimination (identifying which benefits are most preferred).

Results

TBC Baseline Performance and None Coding

We compared two none coding structures: the standard (or "Diener") coding and the alternative naïve none, as previously described. Although both approaches are based on identical survey responses, they differ in how choice data is structured for modeling.

The results were consistent across all three projects: the naïve none approach yielded substantially lower out-of-sample MAEs on the fixed holdout tasks. For example, in Project A, the MAE dropped from 11.90 under standard coding to just 3.33 with naïve none. Projects B and C showed similarly strong improvements.

Image 8 – In Sample Mean Absolute Errors

	Dual Response Coding	**Naïve None**	**Diener None**
Project A	Top Box	1.45	11.70
	70/30 Calibrated	3.16	16.01
	Top 2 Box	3.28	17.12
Project B	Top Box	4.46	15.07
	70/30 Calibrated	2.57	18.97
	Top 2 Box	1.80	12.76
Project C	Top Box	0.19	4.57

	70/30 Calibrated	0.42	4.38
	Top 2 Box	6.09	7.02

Image 9 - Out-of-Sample Mean Absolute Errors

	Dual Response Coding	Naïve None	Diener None
Project A	Top Box	3.33	11.90
	70/30 Calibrated	3.43	16.30
	Top 2 Box	5.80	17.20
Project B	Top Box	5.97	15.00
	70/30 Calibrated	4.32	19.19
	Top 2 Box	5.14	12.77
Project C	Top Box	1.62	4.56
	70/30 Calibrated	1.07	4.64
	Top 2 Box	6.77	7.01

This suggests that the binary structure and consistent task formatting of the naïve none, where every response explicitly informs purchase intent, help the model more precisely estimate individual thresholds, leading to stronger predictive accuracy.

We then assessed whether this coding choice affected the model's ability to rank benefits, a critical function in preference modeling. Using a leave-one-task-out log-likelihood validation, we evaluated how well the model could recover held-out choices across random tasks.

Here, both coding strategies performed similarly. Log-likelihoods were consistent across all three projects, indicating that ranking ability was not compromised by adopting the naïve none approach.

Image 10 – In Sample Benefits Ranking Choice Sets: Sum(Log Likelihoods)/1000 (1st choice set from each screen)

	Dual Response Coding	Naïve None	Diener None
Project A	Top Box	-1.42	-1.42
	70/30 Calibrated	-1.40	-1.40
	Top 2 Box	-1.42	-1.42
Project B	Top Box	-1.38	-1.38

	70/30 Calibrated	-1.36	-1.38
	Top 2 Box	-1.38	-1.38
Project C	Top Box	-2.25	-2.25
	70/30 Calibrated	-2.23	-2.31
	Top 2 Box	-2.27	-2.27

Image 11 – In Sample Naive None Choice Sets: Sum(Log Likelihoods)/1000 (2nd choice set from each screen)

	Dual Response Coding	**Naïve None**	**Diener None**
Project A	Top Box	-0.20	-0.34
	70/30 Calibrated	-0.35	-0.70
	Top 2 Box	-0.14	-0.26
Project B	Top Box	-0.21	-0.34
	70/30 Calibrated	-0.37	-0.73
	Top 2 Box	-0.13	-0.21
Project C	Top Box	-0.39	-0.62
	70/30 Calibrated	-0.52	-1.01
	Top 2 Box	-0.33	-0.52

The naïve none coding structure improved prediction accuracy on fixed tasks, reflected in lower MAEs, without sacrificing the model's ability to rank alternatives. For that reason, all analyses that follow in this paper use the naïve none approach.

Scaling to "Too Many" Attributes: Option 2 and 3 Experiments

Having validated the baseline TBC approach—referred to here as **Option 1**, which uses the full ***n***-choose-***k*** choice task with naïve none coding—we turned to a forward-looking question: *How can we scale TBC to handle even larger feature sets or bundle sizes?*

While Option 1 worked well for up to 40 features, it may become impractical in scenarios where respondents are asked to choose from or build larger bundles (e.g., 10 out of 15 features), due to the exponential growth in the number of possible combinations (15 choose 10 = 3,003 combinations!). To address this, we explored four alternative task structures aimed at either

reducing the number of alternatives shown or restructuring the data to be more digestible for the model.

These task structure variations are summarized below:

- **Option 2A: Append Pairwise Comparison Tasks** – Adds binary comparisons between selected and unselected features.

- **Option 2B: Comparison-Only Tasks** – Drops the original full task and keeps only the binary comparisons.

- **Option 3A: Trimming Unchosen Combinations** – Reduces the number of alternatives by removing bundles made only of unchosen features.

- **Option 3B: Chunking by Chosen Features** – Splits the original choice set into smaller blocks, each centered around a selected feature.

In the following sections, we describe each of these options in more detail and evaluate how well they retain predictive accuracy compared to the baseline.

Option 2A: Append Pairwise Comparison Tasks

This approach augments the original choice data by adding additional tasks that isolate the contribution of each selected feature. After generating the full set of ***n***-choose-***k*** combinations for a given screen, we append ***k*** *new tasks*—one for each chosen feature. In these tasks, each selected feature is directly compared against all features that were *not* selected.

For example, if a respondent chose Features 2 and 4 out of a set of 5, we would add one task comparing Feature 2 to Features 1, 3, and 5, and another comparing Feature 4 to that same set. The intent is to provide the model with clearer, more focused signals: rather than inferring preferences from a single multi-feature choice, we explicitly show that Feature 2 was preferred over 1, 3, and 5—and the same for Feature 4.

Image 12 - Option 2A Coding of a 5 choose 2 task where respondent picked item 2 and 4

Task	Alternative	Item #					Winner
		1	2	3	4	5	
1	1	0	**1**	0	**1**	0	1
1	2	1	1	0	0	0	0
1	3	0	1	1	0	0	0
1	4	0	1	0	0	1	0
1	5	1	0	0	1	0	0
1	6	0	0	1	1	0	0
1	7	0	0	0	1	1	0
1	8	1	0	1	0	0	0
1	9	1	0	0	0	1	0
1	10	0	0	1	0	1	0
2	1	0	**1**	0	0	0	1
2	2	1	0	0	0	0	0
2	3	0	0	1	0	0	0
2	4	0	0	0	0	1	0
3	1	0	0	0	**1**	0	1
3	2	1	0	0	0	0	0
3	3	0	0	1	0	0	0
3	4	0	0	0	0	1	0

Winning Pair

Item 2 (winner) versus item 1, 3, and 5 (losers)

Item 4 (winner) versus item 1, 3, and 5 (losers)

Option 2B: Comparison-Only Tasks (Dropping the Full Combination Task)

This variation takes Option 2A a step further by eliminating the original ***n***-choose-***k*** task entirely. Instead of modeling the full set of possible combinations, Option 2B retains *only* the ***k*** pairwise comparison tasks for each screen.

Using the earlier example—where a respondent selects Features 2 and 4 out of 5—this approach drops the 10-alternative combination task and keeps just the two simpler tasks: one comparing Feature 2 to unchosen features (1, 3, 5), and another doing the same for Feature 4.

The benefit is a significant reduction in complexity: fewer alternatives, smaller task sizes, and potentially faster estimation. The trade-off is that we lose the context of the full joint selection. This approach assumes the individual comparisons are sufficient to capture the underlying preference structure.

Image 13 - Option 2B Coding of a 5 choose 2 task where respondent picked item 2 and 4

Task	Alternative	Item #					Winner
		1	2	3	4	5	
2	1	0	**1**	0	0	0	1
2	2	1	0	0	0	0	0
2	3	0	0	1	0	0	0
2	4	0	0	0	0	1	0
3	1	0	0	0	**1**	0	1
3	2	1	0	0	0	0	0
3	3	0	0	1	0	0	0
3	4	0	0	0	0	1	0

Item 2 (winner) versus item 1, 3, and 5 (losers)

Item 4 (winner) versus item 1, 3, and 5 (losers)

Option 3A: Trimming Unchosen Combinations

While Options 2A and 2B expand the data by adding tasks, Option 3A takes a reductive approach: it simplifies the choice set by removing alternatives made up entirely of non-chosen features. This raises a key question: *Does the model need to see combinations that respondents clearly ignored? Or is it enough to focus on alternatives that include at least one selected item to understand preference signals?*

In this approach, we start with the full ***n***-choose-***k*** set, then drop any alternative that contains *only* unchosen items. For example, if a respondent selects Features 2 and 4 from a set of 5, combinations like {1,3}, {1,5}, and {3,5}—which include none of the selected features—are excluded, reducing the total from 10 to 7.

The trade-off is that we may lose information about how strongly the respondent *dislikes* certain features. That could slightly limit the model's ability to estimate utilities across the full range of preferences. And when ***k*** is close to ***n***, trimming becomes less impactful—there simply aren't many (or any) combinations without a selected item.

Image 14 - Option 3A Coding of a 5 choose 2 task where respondent picked item 2 and 4

Task	Alternative	Item #					Winner Winner
		1	2	3	4	5	
1	1	0	**1**	0	**1**	0	1
1	2	1	1	0	0	0	0
1	3	0	1	1	0	0	0
1	4	0	1	0	0	1	0
1	5	1	0	0	1	0	0
1	6	0	0	1	1	0	0
1	7	0	0	0	1	1	0

← Winning Pair

Option 3B: Chunking by Chosen Features (with added tasks).

While Option 3A trims unchosen-only combinations, the remaining choice sets—especially when ***k*** is large—can still be quite large and complex. Option 3B builds on that idea but focuses on making the data more manageable for the model by breaking up the task into smaller, more focused pieces.

Instead of modeling one large choice set per task, Option 3B splits the data into separate "chunks," one for each chosen item. For example, if a respondent selects Features 2 and 4 out of 5, we create one choice set containing all combinations that include Feature 2, and a second set containing those that include Feature 4. Each chunk includes the original chosen combination plus other pairings of the selected feature with the non-chosen items.

In effect, this turns one 10-alternative task into two smaller, overlapping tasks. It's the same total number of alternatives overall—we're not adding or removing data—but we reorganize it so the model processes preferences in smaller blocks.

Image 15 - Option 3B Coding of a 5 choose 2 task where respondent picked item 2 and 4

Task	Alternative	Item # 1	2	3	4	5	Winner
2	1	0	**1**	0	**1**	0	1
2	2	1	1	0	0	0	0
2	3	0	1	1	0	0	0
2	4	0	1	0	0	1	0
3	1	0	**1**	0	**1**	0	1
3	5	1	0	0	1	0	0
3	6	0	0	1	1	0	0
3	7	0	0	0	1	1	0

All combinations with item 2 (the winner) (Alts 1-4 in task 1)

All combinations with item 4 (the winner) (Alts 1 and 5-7 in task 1)

Running New Models

After recoding our data to reflect each option, we re-estimated models and checked the holdout predictions. The table below summarizes the predictive accuracy (MAEs) for the baseline (Option 1) versus these new options. The green highlight is the best performer in the row and the red highlight is the worst performer in the row.

Image 16 - In Sample Mean Absolute Errors

	Dual Response Coding (Naïve None)	**Option 1** Baseline	**Option 2A** Append Pairwise Comparisons	**Option 2B** Comparison-Only Tasks	**Option 3A** Trimming Unchosen Combinations	**Option 3B** Chunking by Chosen Features
Project A	Top Box	1.45	7.70	5.31	1.19	9.10
	70/30 Calibrated	3.16	10.89	5.61	3.11	14.27
	Top 2 Box	3.28	9.96	6.97	3.17	11.73
Project B	Top Box	4.46	11.00	8.42	4.14	12.34
	70/30 Calibrated	2.57	14.58	8.73	2.34	18.15
	Top 2 Box	1.80	7.60	4.98	1.62	8.73
Project C	Top Box	0.19	5.93	1.27	0.20	5.43
	70/30 Calibrated	0.42	9.53	1.33	0.56	10.28
	Top 2 Box	6.09	3.96	3.69	6.77	4.12

Image 17 - Out of Sample Mean Absolute Errors

	Dual Response Coding (Naïve None)	Option 1 Baseline	Option 2A Append Pairwise Comparisons	Option 2B Comparison-Only Tasks	Option 3A Trimming Unchosen Combinations	Option 3B Chunking by Chosen Features
Project A	Top Box	3.33	7.66	5.32	3.27	9.03
	70/30 Calibrated	3.43	10.74	5.82	3.33	14.16
	Top 2 Box	5.80	9.81	7.61	5.76	11.57
Project B	Top Box	5.97	10.95	8.77	5.88	12.37
	70/30 Calibrated	4.32	14.28	8.85	4.00	17.92
	Top 2 Box	5.14	8.73	6.70	5.09	9.71
Project C	Top Box	1.62	5.49	2.06	1.47	5.34
	70/30 Calibrated	1.07	8.88	1.56	1.06	9.62
	Top 2 Box	6.77	4.48	3.90	7.66	4.70

The initial findings were intriguing. Option 3A (Trimming Unchosen Combinations) performed relatively well – its MAE was on par with or slightly better than the Option 1 baseline in some cases. Removing the "loser-only" alternatives didn't degrade performance and even seemed to help. This suggests that, at least in our data, little information was lost by cutting out combinations of unchosen features.

In contrast, Options 2A, 2B, and 3B underperformed. Their MAEs were significantly higher (worse) than the Option 1 baseline . This was true both in-sample and out-of-sample. In other words, simply adding a bunch of comparisons (2A) or splitting tasks (3B), without any other adjustments, actually hurt the model's ability to predict the holdouts. We also confirmed via log-likelihood analyses that these options were doing a poorer job at reproducing respondents' choices compared to the baseline.

At first glance, the results may seem counterintuitive. In Options 2A and 3B, we gave the model *more* data—so why did performance decline? And in 2B, we simplified the task structure—shouldn't reducing complexity help, or at least not hurt? Part of the answer might be in how hierarchical Bayes models interpret and weigh the data they're given.

Overfitting and Shrinkage

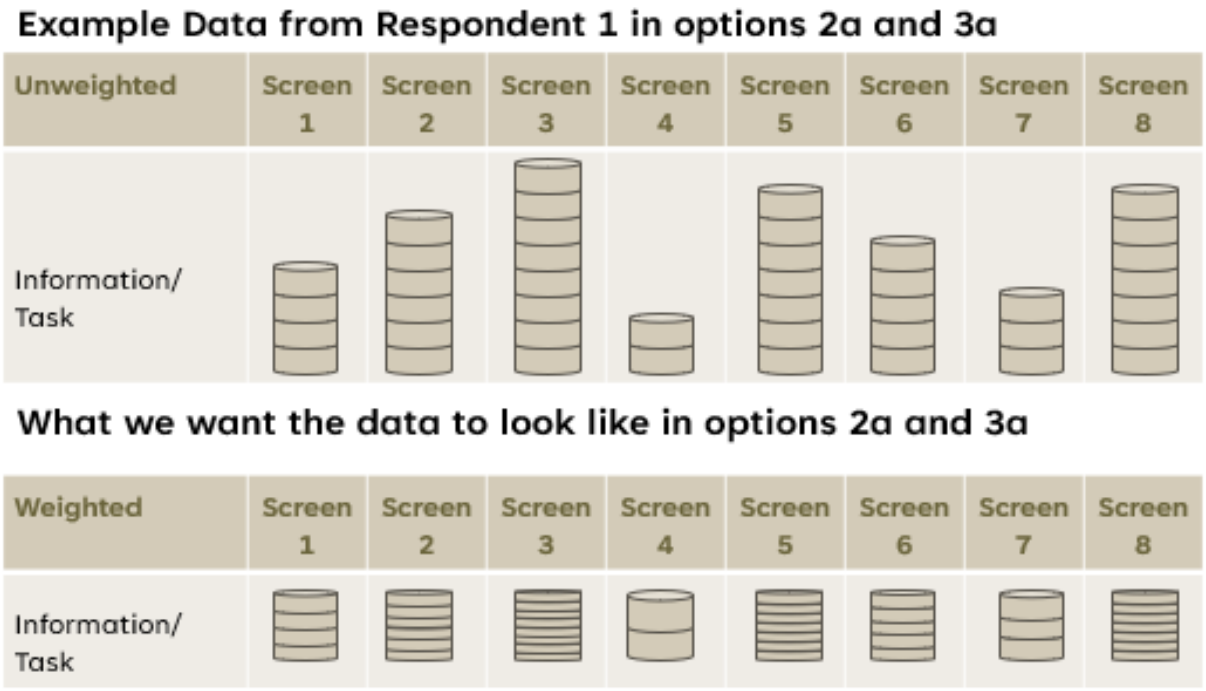
In an HB model, each respondent's part-worths are influenced by two forces: the individual's own data (likelihood) and the population distribution (prior). When a respondent has a lot more data points, the model trusts their data more and shrinks their utilities less toward the mean. In Options 2A, 2B, and 3B, each respondent's number of "observations" is effectively increased (because we added extra derived tasks for each of their original tasks). But those extra observations are not providing truly independent new information – they are derived from the same choice. The model doesn't know that, so it treats them like additional evidence and gives less weight to the prior. In short, we accidentally overfit each individual by feeding the model redundant data. The HB posterior for each person became more confident (less shrinkage), which can hurt out-of-sample prediction if those people had any noise in their choices (and everyone has some). This phenomenon can cause the overall fit (especially out-of-sample) to worsen, as we saw.

One challenge with Options 2A, 2B, and 3B is that they introduce imbalance in how much data each screen contributes, depending on the number of items selected (***k***). When ***k*** = *1*, only one comparison task is created. But when ***k*** = *6*, that same screen might produce six separate tasks—one for each selected item compared against the rest.

This means that screens with higher ***k*** generate significantly more data, even though it's not clear they provide proportionally more information. Choosing one item (***k*** = 1) reflects a clear top preference. But selecting six out of seven items (***k*** = 6) is closer to identifying the *least* important item—which may not be six times more informative. The result is an uneven data structure that can bias the model toward overfitting high-***k*** screens, while underweighting the sharper signals from low-***k*** ones.

Given these issues, we wondered whether applying a weighted likelihood could address both problems: the over-counting of data from added or chunked tasks, and the imbalance introduced by varying ***k***. By adjusting the weight of each task based on how it was generated, we aimed to restore balance and prevent any one screen from dominating the model.

Image 18 - A visual of the “problem” of imbalance with low-k vs. high-k screen



Weighted Analysis: Fixing the Balance

To address the imbalance, we modified our HB estimation to apply weighted likelihoods to the synthetic tasks. The goal was to ensure that each original screen contributes equally to the model, regardless of how many tasks were generated from it.

For example, if a screen produced k additional tasks (as in Option 2A), we assigned a weight of $1 / (k+1)$ to each of the $k+1$ tasks—including the original. This kept the total contribution from that screen equal to 1, preserving parity across all screens.

Option 3A didn’t introduce any new tasks; it only reduced the number of alternatives per task. As a result, no weighting adjustment was needed—its contribution remained balanced by design.

We re-estimated the models for Options 2A, 2B, and 3B using the weighted likelihood approach—and the results were striking. In-sample MAEs dropped significantly, in some cases even outperforming the Option 1 baseline. More importantly, out-of-sample MAEs improved across the board, matching or exceeding the baseline’s performance.

Image 19 - In Sample Mean Absolute Errors with Weighted Likelihoods

	Dual Response Coding (Naïve None)	Option 1 Baseline	Option 3A Trimming Unchosen Combinations	Option 2A Append Pairwise Comparisons Weighted	Option 2B Comparison-Only Tasks Weighted	Option 3B Chunking by Chosen Features Weighted
Project A	Top Box	1.45	1.19	1.06	1.21	0.99
	70/30 Calibrated	3.16	3.11	2.96	3.10	3.16
	Top 2 Box	3.28	3.17	2.45	2.41	2.90
Project B	Top Box	4.46	4.14	1.80	1.61	3.47
	70/30 Calibrated	2.57	2.34	2.17	2.02	2.16
	Top 2 Box	1.80	1.62	0.71	0.62	1.12
Project C	Top Box	0.19	0.20	2.03	3.09	0.58
	70/30 Calibrated	0.42	0.56	0.25	0.26	0.45
	Top 2 Box	6.09	6.77	9.16	10.15	7.85

Image 20 - Out of Sample Mean Absolute Errors with Weighted Likelihoods

	Dual Response Coding (Naïve None)	Option 1 Baseline	Option 3A Trimming Unchosen Combinations	Option 2A Append Pairwise Comparisons Weighted	Option 2B Comparison-Only Tasks Weighted	Option 3B Chunking by Chosen Features Weighted
Project A	Top Box	3.33	3.27	3.38	3.53	3.20
	70/30 Calibrated	3.43	3.33	3.75	4.02	3.52
	Top 2 Box	5.80	5.76	5.23	5.10	5.56
Project B	Top Box	5.97	5.88	5.40	5.32	5.64
	70/30 Calibrated	4.32	4.00	3.13	3.15	3.57
	Top 2 Box	5.14	5.09	4.58	4.39	4.89
Project C	Top Box	1.62	1.47	1.87	2.73	1.41
	70/30 Calibrated	1.07	1.06	1.18	1.02	1.13
	Top 2 Box	6.77	7.66	9.08	9.90	8.59

Notably, Option 2B—which had one of the worst unweighted prediction errors—became the best-performing model when weighted, delivering the lowest out-of-sample MAE among all conditions. Weighted versions of 2A and 3B also rebounded, performing on par with or slightly better than the baseline.

In short, weighting resolved the overfitting problem: it allowed the additional comparisons to enhance the model without overpowering it.

The likelihood-based validation supported these findings. Without weighting, models like Options 2B and 3B showed substantially lower log-likelihoods than the baseline, indicating they were less effective at ranking alternatives—especially in scenarios involving a "None" option. But once we applied weighted likelihoods, performance improved dramatically: the weighted versions of 2B and others matched or even exceeded the baseline, particularly in predicting purchase decisions.

Image 21 – In Sample Benefits Ranking Choice Sets: Sum(Log Likelihoods)/1000

		Naïve None					**Diener None**	
	DR Coding	**Opt 1**	**Opt 3A**	**Opt 2A weighted**	**Opt 2B weighted**	**Opt 3B weighted**	**Opt 1**	**Opt 1 Weighted**
Project A	T1B	-1.42	-1.41	-1.41	-1.41	-1.41	-1.42	-1.41
	Calibrated	-1.40	-1.39	-1.40	-1.41	-1.40	-1.40	-1.41
	T2B	-1.42	-1.41	-1.41	-1.41	-1.41	-1.42	-1.41
Project B	T1B	-1.38	-1.37	-1.37	-1.38	-1.37	-1.38	-1.37
	Calibrated	-1.36	-1.36	-1.38	-1.39	-1.36	-1.38	-1.38
	T2B	-1.38	-1.37	-1.37	-1.38	-1.37	-1.38	-1.37
Project C	T1B	-2.25	-2.25	-2.31	-2.34	-2.26	-2.25	-2.29
	Calibrated	-2.23	-2.25	-2.33	-2.36	-2.27	-2.31	-2.36
	T2B	-2.27	-2.26	-2.31	-2.34	-2.27	-2.27	-2.27

Image 22 – In Sample Naive None Choice Sets: Sum(Log Likelihoods)/1000

	Naïve None	**Diener None**

	DR Coding	Opt 1	Opt 3A	Opt 2A weighted	Opt 2B weighted	Opt 3B weighted	Opt 1	Opt 1 Weighted
Project A	T1B	-0.20	-0.19	-0.18	-0.18	-0.19	-0.34	-0.27
	Calibrated	-0.35	-0.34	-0.32	-0.32	-0.34	-0.70	-0.68
	T2B	-0.14	-0.14	-0.13	-0.13	-0.14	-0.26	-0.19
Project B	T1B	-0.21	-0.20	-0.18	-0.18	-0.20	-0.34	-0.27
	Calibrated	-0.37	-0.36	-0.34	-0.33	-0.35	-0.73	-0.72
	T2B	-0.13	-0.12	-0.12	-0.12	-0.12	-0.21	-0.16
Project C	T1B	-0.39	-0.37	-0.32	-0.31	-0.35	-0.62	-0.49
	Calibrated	-0.52	-0.50	-0.46	-0.45	-0.48	-1.01	-0.98
	T2B	-0.33	-0.31	-0.27	-0.27	-0.29	-0.52	-0.40

Conclusions and Key Learnings

Token-Based Conjoint (TBC) offers a practical and scalable solution to the long-standing challenge of evaluating large sets of product features. Across multiple studies, it proved effective at handling up to 40 attributes while maintaining both respondent engagement and analytical robustness.

Two innovations were important to TBC's success:

- The naïve none coding improved purchase prediction by consistently modeling the respondent's threshold across all tasks.
- Weighted likelihoods corrected for overrepresentation when tasks were added or chunked, preserving model balance and accuracy.

We also demonstrated that TBC can be extended beyond its original format. Option 3A (trimming unchosen-only alternatives) and Options 2B/3B with weighting (which split or simplify tasks) offer promising paths for scaling TBC to even more complex research designs.

When and Why to Use TBC

TBC is especially well-suited to studies with 20+ attributes, where traditional conjoint designs become too complex and MaxDiff lacks the additive structure required for bundle simulation. Compared to CBC, TBC focuses more on feature prioritization than direct trade-offs between

complete product profiles. And unlike MaxDiff, it enables utility-based purchase simulations while still keeping the task lightweight for respondents.

From a respondent perspective, TBC was intuitive and engaging. Participants "built" their ideal product step-by-step and reported purchase likelihood, creating a more natural experience than completing back-to-back CBC and MaxDiff modules. From a stakeholder perspective, the outputs were straightforward and actionable: top feature rankings, uptake simulations, and clear diagnostics on how many features are needed to make a compelling offer.

However, TBC is not without limitations:

- Comparisons between bundle sizes (i.e., different values of k) are not directly observed. Any inferences across k rely on extrapolation via transitivity through the "None" response.

- Data is sparse for low-ranked features, so utility estimates for consistently unchosen items are less precise. Researchers may need to supplement TBC with targeted tasks or fixed evaluations if insights into bottom-tier features are critical.

- Interpretability can decline as modeling complexity increases. Techniques like synthetic task generation and weighting improve performance, but can make the model harder to explain to non-technical audiences.

- Performance beyond 40+ features remains untested. While early signs are promising, we have not yet fielded live studies with 60, 80, or more attributes.

Call to Action

TBC opens new possibilities for high-dimensional product research, but there is much to explore. As studies scale to 60+ features, adaptive designs—such as those inspired by Bandit MaxDiff—could be used to intelligently oversample features until their utility upper bounds are known.

Beyond scalability, future versions of TBC could benefit from modeling utility as a non-linear function of part-worths, rather than assuming simple additivity. This would allow for explicit diminishing returns as more items are added, better capturing how people experience bundles and helping prevent overstated uptake from purely additive models.

Finally, pricing remains an open area for development within the TBC framework. To date, no formal work has explored how to incorporate price directly into token allocation or task design. Future research is needed to understand whether this is possible.

Last Words

In summary, TBC offers a powerful and respondent-friendly approach for understanding feature-level preferences in high-dimensional settings. By combining the strengths of MaxDiff and CBC with a simple yet informative token allocation mechanic and follow-up purchase intent, TBC enables deeper exploration without overwhelming respondents.

That said, TBC isn't universally the right tool. Its strengths shine when pricing isn't central—such as when prices are already fixed or the focus is on additive value. In contrast, if the aim is to include price as an attribute and optimize for revenue, we advise caution; methods like CBC or a MaxDiff + CBC hybrid may be more appropriate.

Ultimately, the choice of method should reflect what the client is trying to learn. Each approach comes with trade-offs. TBC has a valuable place in the toolkit—especially when tackling high-dimensional problems where traditional conjoint methods struggle. With thoughtful application, it can help teams prioritize features intelligently and design products that resonate.

References

Brazell, J., C. Diener, E. Karniouchina, W. Moore, V. Séverin and P. Uldry 2006. The no-choice option and dual response choice design. *Marketing Letters,* 17:255-268.

Diener, C., B. Orme and D. Yardley 2006. Dual response "none" approaches: theory and practice. In *Sawtooth Software Conference Proceedings,* pp. 157-168. Sequim, Wash.: Sawtooth Software.

Chrzan, K. and B. Orme. 2017. Becoming an expert in conjoint analysis. Sawtooth Software.

Lattery, K. 2016. Dual response rating scale analysis. Paper presented at the Turbo Choice Modeling Event, Captiva Island, Fla.