Market Optimization Engine

Architectural Overview

Holden Coffman

June 2025

"To optimize a two-sided market, you need more than just a great matching engine – you need a unified, cohesive optimization platform that intelligently anticipates & balances supply and demand."

Executive Summary

The following paper outlines the conceptual architecture of a market optimization engine designed for the Facilities Maintenance industry. Inspired by the pricing systems that power companies like Uber, Airbnb, and Amazon Flex, this framework seeks to bring similar optimization logic to a space that has traditionally relied on static contracts, human heuristics, and lagging indicators.

The core idea is simple but powerful: profit is driven not just by the prices we charge customers or pay providers, but by how those two prices interact in a dynamic two-sided marketplace to shape volume, conversion, and margin. This interaction can be modeled. It can be simulated. And it can be optimized.

At the heart of the engine is an objective function — a symbolic equation that defines profit as a function of customer pricing, provider pricing, and predicted win rates on both sides of the market. Logistic regression models are employed as an initial heuristic to estimate win probabilities, with the long-term goal of replacing them with more advanced ML models such as XGBoost. A bridge layer — the Volume-Feasibility Link (VFL) — serves as the adapter between customer-side projections and provider-side fill probabilities, enabling the simulation of seasonal job flows and their fulfillment rates at different pricing levels.

What emerges is a dynamic pricing loop: one that can test thousands of customer pricing scenarios, simulate provider response behavior to each, and identify the rates that maximize profit, market share, or any objective we choose to prioritize. And because the logic is structured, not hardcoded, the architecture is flexible — supporting everything from basic margin maximization to more advanced strategies like fulfillment time minimization, client retention, or cost containment.

This system is still early in its lifecycle, but the scaffolding is now in place. The vision is clear. Each layer of the architecture is deliberately designed to support the next. And critically, it does not compete with the platform initiatives already underway — it completes them. What [*redacted*]'s team is building on the job-matching side is essential. But without a unifying optimization layer to govern pricing and fulfillment strategy, we will inevitably face the same scaling limits again. This proposal offers that missing layer: an economic intelligence engine that ensures every job we win is the right job — priced right, fulfilled smartly, and leveraged for long-term market advantage.

Table of Contents

Note: All exhibits appear in the Appendix beginning on page 15

I. Introduction

This paper is meant to be an architectural sketch of the overall optimization engine I am seeking to design and implement for the Facilities Maintenance industry. It is inspired by companies who have succeeded in applying theoretical economics at scale by implementing machine learning trained on market data (e.g. Uber, Airbnb, Amazon Flex). The objective function itself is something that came to me organically after being tasked with modelling customer price sensitivity, when I began to wonder how a customer price sensitivity model would ever result in actual suggested pricing (the model only tells you how the customer will respond to pricing, not what price you should choose). It was at that point that I began to develop the idea of a decision-making framework that sits above the model, and uses its output as part of a broader structure in order to make strategic decisions based on company goals, leveraging the model's insight into customer behavior to simulate scenarios and choose the best outcome. Very quickly the concept of a profit function crystallized in my mind, which would draw on the output of the customer price sensitivity model in order to simulate expected profit over a continuum of potential hourly rates. But the next problem was how we would model cost - and this led to the realization that we would also need to model provider behavior in much the same way we were planning to model customer behavior, in order to capture trends and predict provider outcomes.

As these thoughts progressed, they gave birth to the **objective function** which is really the core of this entire project's vision. The objective function - which is our North Star - is explained below, with subsequent sections explaining each of its fundamental inputs as well as the bridge logic that ties the inputs together and allows us to predict optimal (profit-maximizing) pricing.

This general architecture, as I have been discovering, is not novel - though it has not yet been implemented in many industries. It is a scaffolding we will set from the beginning like a blueprint, filling in each component with a view to how it interacts with the others and how they ultimately all work together to allow the broader optimization engine they fit into to run effectively. This is exactly the kind of optimization engine that is being utilized successfully at companies like Uber, and the exciting prospect for us right now is that there are no major players in the Facilities Maintenance industry that have implemented anything like it yet. The upside potential is **extreme**.

II. Objective Function

Overview

The objective function is the heart of the “optimization engine”. This is the equation we feed the output of each model into, in order to simulate expected profit at any particular customer rate. From there, we solve for the maximum point and ask what customer rate/provider rate combination exists at that point. The objective function takes the shape of a very simple formula:

Profit = Win rate * (Customer Hourly Rate - Provider Hourly Rate)

This is the basic architecture - the simple framework with which we began. Three basic inputs:

1. Win Rate: How much work do we expect to win?
2. Customer Hourly Rate: What is the rate that must be offered to the customer to win that amount of work?
3. Provider Hourly Rate: What rate must be offered to providers to fill that amount of work?

It’s a very simple scaffolding, but it is the same starting point that other successful market optimizing engines began with. It gives us a framework that we can fill out with sophisticated ML modeling techniques and macroeconomic principles in order to visualize and simulate market trends so that we can price **proactively**.

Starting Simply: Deriving the Objective Function Using Logistic Functions

Our initial customer price sensitivity model was built using logistic regression. This was very helpful, as it allowed us to represent the model using familiar symbolic logic: in other words, a formula that we could apply algebra to.

Once our logistic regression was trained, we had a simple formula we could use to represent customer win rate as a function of customer hourly rate. It wasn’t a very far leap to imagine a similar model trained on provider data that would represent provider win rate as a function of provider hourly rate. From here the question was: how would we establish a relationship between the two functions in order to feed them up into the profit (objective) function? The idea at this point was obvious: **set the win rates equal to one another**. The assumption was that, if we picture the customer and provider markets to possess the same total volume of work, it would require the same overall win rate on the provider side to fill the customer work that you won on the customer side (this basic assumption will be refined later…it was just a starting point to sketch the concept).

From here, it was simply a matter of setting the two logistic functions equal to each other (equal win rates means equal outputs/win rates), and solving for provider hourly rate as a function of customer hourly rate. Now, you have both of your functions reduced down to one single variable: customer hourly rate. All inputs in the profit function (win rate, customer hourly rate, and provider hourly rate) could be expressed in terms of that single variable. You now have a unified profit function expressed in terms of logistic formulas that can be solved for a max point, thus guiding you to the customer and provider hourly rate pair which yields the optimal outcome.[1]

In this framework, customer pricing leads the scenario simulation - as it should. Everything starts with the RFP. The customer hourly rate we decide to present determines the win rate, which determines the needed provider quantity of work, which determines what rate you'll need to offer providers to fill that quantity of work, which then all combines together to determine what margin you'll get on average, which when combined with how much work you expect to win and fill, tells you expected gross profit.

Moving Forward: Future Iterations Employ the Same Mental Model

At this point in the process, we have the mental model/scaffolding securely in place. As we progress we can swap in much more complex and effective machine learning models than logistic regression to map customer and provider behavior (e.g. gradient boosted decision trees). The reason this works is that, although these more complex models cannot be represented using symbolic logic like a logistic function, the conceptual architecture of the objective function will be the same:

1. Model customer price sensitivity
2. Model provider price sensitivity
3. Establish a link between the two models' outputs based on quantity of work
4. Feed the model's inputs into the objective function in order to simulate all possible outcomes and choose the outcome which maximizes gross profit

[1] Please see Exhibit A in the appendix section for a detailed walkthrough of the initial profit function's derivation using logistic functions. It provides a helpful heuristic to anchor into the core components of the mental model.

The main idea here is this: **we now have the objective function's underlying structure established**. Future iterations of each model (customer and provider) will interact in the same fundamental ways, though the models themselves and the challenge of linking their outputs will become much more complex.

The task of the rest of this paper is to outline how we scale from the handcrafted heuristics above to an advanced optimization pipeline. But we have the vision in place: we know where we're headed.

III. Customer Model - The Demand Curve

Modeling Customer Price Sensitivity

The initial prototype of the customer price sensitivity model was developed using scikit-learn and logistic regression, trained on a binary classification dataset to predict win/loss outcomes as a function of customer hourly rate, trade, and location. This model was a simple proof-of-concept built on highly generalized RFP data (often state-level contracts that don't necessarily correlate to customer spend), but we are working to source the required data to track revenue realization on a store-by-store level for our next model.

The next iteration of the customer price sensitivity model will be a gradient boosted decision tree model from the XGBoost library in Python. Here is a quick sketch of the model's blueprint as of today:

- Features
    - Latitude
    - Longitude
    - Customer
    - Customer Industry
    - Customer Hourly Rate
    - Service Trade
    - Total Addressable Spend Per Store For That Trade
    - Customer's Total Revenue
- Output (Target Variable)
    - Gross Revenue Realized
- Granularity
    - Per Year Per Store

The model will be trained on a dataset containing the features and outputs listed above, in order to learn to map the relationship between revenue realization and each feature. The result will be a regression model that can accurately predict revenue realization at the annualized store level as a function of customer hourly rate, service trade, customer industry, customer size, location, and other features.

IV. Provider Model - The Supply Curve

Modeling Provider Price Sensitivity

This model would be built with similar features to the current labor rate model; however the target variable would be the likelihood of job acceptance. This means that we would use a target variable such as binary 0/1 for job acceptance, or possibly time to fill as a fraction of the total posting window. The key point is that we would want to be able to model the likelihood of a job being filled within the desired cutoff time for any given rate.

Once we had a model of this nature built, we would have visibility into the market's supply curve, thus providing us with the crucial market intelligence we need to simulate provider response to an entire continuum of different pricing scenarios. To put it succinctly: training a model to predict likelihood of provider acceptance allows us to model the supply curve and gain critical knowledge of supply elasticity, rather than locking us into historical pricing patterns. It gives us real-time visibility **not just into what rates providers have historically accepted, but how much lower of a rate they would have accepted**. This is what allows true optimization scenarios to be run so that we can price proactively with agility.

At a high level what we are suggesting is this:[2]

- We would model provider price sensitivity *prescriptively rather than descriptively:* not by predicting historical accepted rates, but by modeling probability of job fulfillment as a function of price.
- The requisite data to construct our target variable likely already exists within DMG's databases. Data like first posting acceptance/rejection could be used to train a classification model, or time to fill could be used to train a regression or probabilistic regression.
- This allows us to simulate how different pricing scenarios affect expected volume, and feed that into a broader optimization loop with the customer-side model.

[2] Please see Exhibit B in the appendix section for a more detailed commentary on the provider labor rate model that would be necessary for this optimization engine.

V. Bridge Layer: The Volume-Feasibility Link

The Need for A Bridge Layer Explained

One of the key challenges in this optimization problem is the fundamental difference in how customer and provider interactions take place. Customer negotiations take place at an annualized level per store per trade. Provider negotiations take place daily on a job-by-job basis. This difference in granularity between the two models forms a challenging difficulty around how the two models' outputs will be linked. We will need a bridge layer – like an adapter between the two models – that translates the output of one into terms which are consistent with the structure of the other.

A Path Forward: The Volume-Feasibility Link[3]

Volume-Feasibility Link (VFL) is the name we have settled on for an interactive layer that would bridge outputs between the customer and provider models. Essentially the flow of information would be:

- Customer model → VFL: Anticipated annual revenue for a store based on chosen customer hourly rate
- VFL simulates an entire year's worth of jobs based on that total revenue value: each with their own features around urgency, provider depth, recent work volume, etc. - features that would vary throughout the season
- VFL → Provider Model: All jobs for the season are sent to the provider model, which predicts the needed rate to reach 100% fill probability on each simulated job
- Provider Model → VFL: Needed rates are fed back to the VFL, which calculates total seasonal cost of all jobs, and passes the result up to the objective function

Note: Provider Model Would Be Used Upstream and Downstream

One important note at this point is that, although in this optimization framework the provider model would be used "upstream" - i.e. on simulated data well in advance of any actual work orders taking place, the same model **could and should** be deployed

[3] Please see Exhibit C in the appendix section for a visualization of the Volume-Feasibility Link's information flow.

“downstream” - i.e. during the subsequent season to power the real-time provider pricing that is posted in the app.

This is actually a crucial implementation requirement, because we would need to be pricing downstream during the season with the same model that we used to simulate pricing upstream, prior to the season. This is for a couple of reasons:

1. Actual in-season pricing, when powered by an elasticity-based model, will be far more optimal. The model will guide each job to the truly minimum viable rate at which it can be effectively filled, with granular job-by-job level market intelligence.
2. Using the same model for pre-season predictions and in-season actuals forms a needed feedback loop which helps the VFL gather consistent data and improve its simulating accuracy over time.

VI. Optimization Loop

At this point, each of the fundamental components of the optimization engine have been sketched out. It only remains to explain how they form a cohesive whole, interacting to drive company decision-making toward market-optimizing outcomes. This takes place through an optimization loop process commonly known as **grid search**.

Grid Search Method

Step 1: A potential hourly rate is fed into the customer price sensitivity model, along with the other features appropriate to that store. The model predicts gross revenue won at that rate

Step 2: Gross revenue for the year is fed to the VFL, which simulates a year's worth of jobs and passes those jobs down to the provider price sensitivity model

Step 3: The provider model estimates the lowest hourly rates needed to ensure fulfillment of each job within quality/speed constraints, and feeds these back to the VFL

Step 4: The VFL applies the rates generated by the provider model to the size of each simulated job in order to determine gross cost, summing output together for the entire year

Step 5: Each input is fed into the objective function (gross revenue and gross cost) in order to calculate expected profit for the year

Step 6: This process is looped over an entire continuum of potential customer rates (e.g. $0/hr to $1000/hr) in order to calculate expected profit at each rate and select the rate that yields the highest outcome

This process is the flow of logic which forms the market optimization engine. It is the exact kind of architecture that companies like Uber and Airbnb have implemented to profitably optimize dynamic, two-sided marketplaces through sophisticated insight into the supply and demand curves within their industries.[4]

We now have:

- A dynamic pricing engine that chooses rates **based on how both sides of the market behave.**

[4] Please see Exhibit D in the appendix section for a visual illustration of the optimization engine's architecture.

- Real-time optimization: instead of just perpetuating historical rates, or posting historical rates with a slight haircut based on general market trends, the system is dynamically selecting rates to balance the supply and demand curves in order to optimize outcomes at a granular level.
- A real AI pricing system that can continue to evolve with more data and smarter objectives.

VII. Conclusion and Roadmap

The vision presented in this paper is not a finished product - it’s a scaffolding. What we have built here is a foundational architecture, a mental model that can be incrementally improved as data grows, modeling techniques evolve, and strategic priorities shift. This flexibility is one of its greatest strengths: **each layer of this architecture can be refined or swapped out without losing the coherence of the overall optimization engine**.

Modern ML workflows offer powerful ways to accelerate these improvements. Leveraging large language models (LLMs) like GPT-4 in the modeling process can assist with feature selection and engineering, flagging data leakage, tuning hyperparameters, interpreting model performance, and even automating parts of the pipeline. This means that **the complexity of implementing this system can be lowered dramatically, enabling us to prototype quickly and improve continuously**.

Importantly, this framework also opens the door to a future state where customer pricing becomes as democratized as provider pricing. Right now, provider pricing operates at the job level, while customer pricing decisions are made through static, state/store-level contracts that cover the entire year. This optimization engine creates a path to democratize job-level decision-making across both sides of the market - ultimately enabling the platform to function more like Uber or Airbnb, where real-time price discovery and elastic behavior shape every transaction. Our technology could even play a role in moving the industry toward that future, as customers gravitate toward faster, higher-quality fulfillment at more competitive prices - driving widespread adoption of our app and shifting behavior away from traditional contract-based arrangements. Over time, this could reposition our company not just as a service coordinator, but as the central intelligence layer of the facilities maintenance economy - powering real-time pricing, fulfillment, and market coordination at scale. This is an ambitious vision, but it has already been realized in other industries.

Another important feature of this framework is that it's objective-agnostic. **The structure of the optimization engine supports a wide variety of business goals**. While this paper focuses on profit maximization, the same loop could easily be repurposed to prioritize objectives such as market share growth, cost containment, fulfillment speed, or client retention - simply by adjusting the objective function. The inputs remain the same, only the output criteria shift. This flexibility makes the architecture adaptable to evolving strategies over time.

Once we have two strong models that predict market behavior on both sides (customer and provider), and a bridge layer that links their outputs, the rest of the engine becomes a flexible simulation framework. **From there, optimization becomes a matter of asking the right question—and letting the engine find the right answer.**

Appendix

Exhibit A: Derivation of the Objective Function

Derivation of the Objective Function

Using Logistic Functions to Mentally Model an Optimization Framework

---

Step 1: Define Profit in Our Context

We begin by expressing the goal of our objective function: profit optimization. In its most basic form, profit is defined as:

$$P = R - C$$

Where:

- P is profit
- R is revenue
- C is cost

Revenue is the expected revenue from a job, which we define as:

$$R = W_c \cdot x_c$$

Where:

- W_c is the customer win rate (probability of acceptance)
- x_c is the proposed customer hourly rate

Cost is the expected cost to fulfill the work:

$$C = W_p \cdot x_p$$

Where:

- W_p is the provider win rate (probability of acceptance)
- x_p is the provider hourly rate required to achieve W_p

Substituting these into the profit formula yields:

$$P = W_c \cdot x_c - W_p \cdot x_p$$

*Note that, for the sake of simplicity, quantity is excluded from the revenue and cost formulas here (really what we are depicting is profit per hour of work, which can be easily multiplied by expected labor hours to yield gross profit).

Step 2: Model Win Rates Using Logistic Regression

We now use logistic regression to model both W_c and W_p , which describe the price sensitivity/elasticity curves of both supply and demand.

Customer Win Rate Model:

$$W_c = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_c)}}$$

Provider Win Rate Model:

$$W_p = \frac{1}{1 + e^{-(\beta_A + \beta_B x_p)}}$$

Where:

- $\beta_0, \beta_1, \beta_A, \beta_B$ are static model coefficients
- x_c is customer hourly rate and x_p is provider hourly rate

Step 3: Establish A Relationship Between Customer and Provider Rates

Premise: For any given amount of work that is awarded on the customer side, we must win the same amount of work on the provider side in order to meet demand.

Conclusion: If supply must meet demand, we can set the two win rates equal to each other, allowing us to solve for provider rate as a function of customer rate. Observe:

$$W_c = W_p$$

Using the logistic functions expressed in Step 2:

$$\frac{1}{1 + e^{-(\beta_0 + \beta_1 x_c)}} = \frac{1}{1 + e^{-(\beta_A + \beta_B x_p)}}$$

Solving algebraically:

$$\beta_0 + \beta_1 x_c = \beta_A + \beta_B x_p$$

$$\Rightarrow x_p = \frac{\beta_0 + \beta_1 x_c - \beta_A}{\beta_B}$$

We now have **provider rate as a function of customer rate**.

Step 4: Derive Final Profit Function

Return to the original profit function:

$$P = W_c x_c - W_p x_p$$

Since we've established that $W_c = W_p$, we can simplify this to:

$$P = W_c(x_c - x_p)$$

Now substitute in the logistic expression for W_c and the derived expression for x_p :

$$P = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_c)}} \cdot \left(x_c - \frac{\beta_0 + \beta_1 x_c - \beta_A}{\beta_B} \right)$$

Simplify:

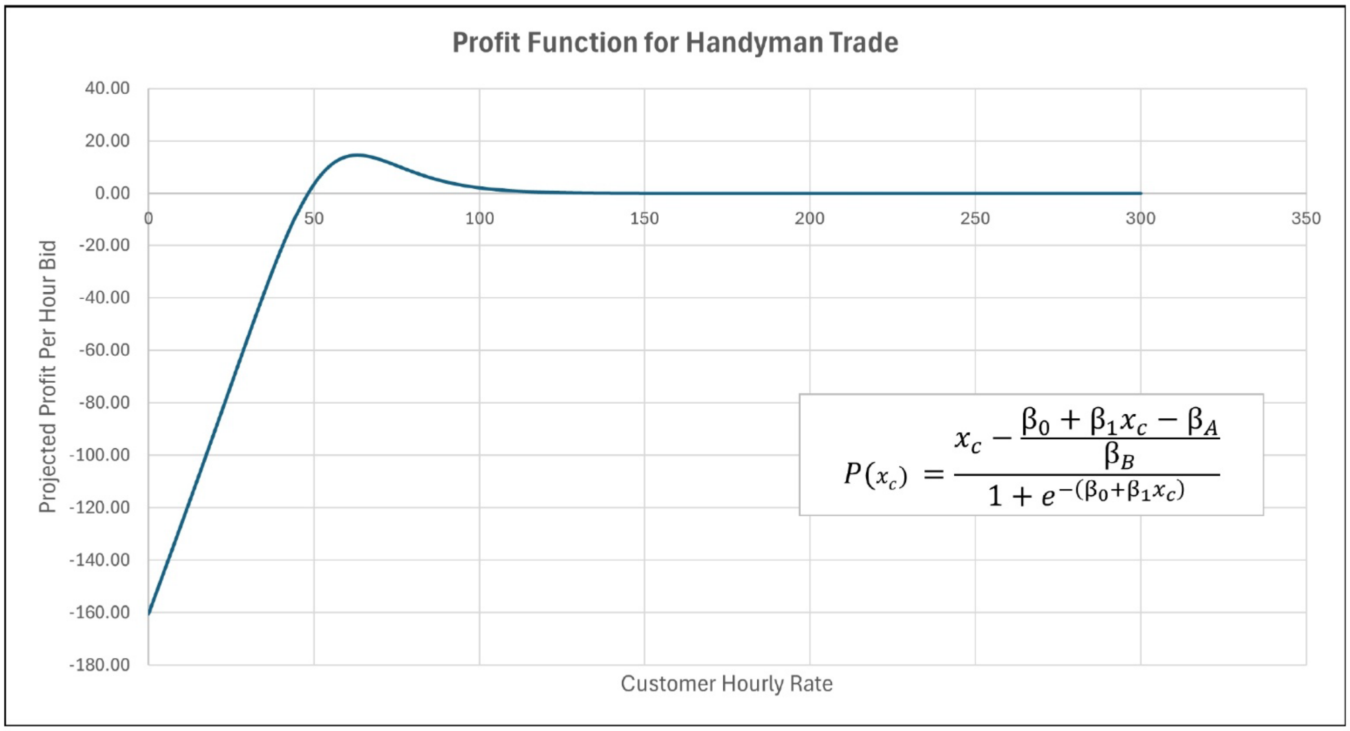
$$P = \frac{x_c - \frac{\beta_0 + \beta_1 x_c - \beta_A}{\beta_B}}{1 + e^{-(\beta_0 + \beta_1 x_c)}}$$

Interpretation

We have now derived a symbolic profit function expressed in terms of a **single decision variable**: the customer hourly rate x_c . This is a powerful result, as it enables two-dimensional optimization and visualization of expected profit across different pricing scenarios.

This derivation lays the conceptual foundation for more complex ML-powered optimization layers to be built atop — enabling future iterations that replace symbolic logistic forms with non-parametric models like gradient boosted trees, while maintaining the same architectural scaffolding.

Profit Function Visualized Graphically



$$P(x_c) = \frac{x_c - \frac{\beta_0 + \beta_1 x_c - \beta_A}{\beta_B}}{1 + e^{-(\beta_0 + \beta_1 x_c)}}$$

Exhibit B: Limitations of the Existing Provider Labor Rate Model

In order to execute on the optimization architecture proposed in this paper, we would need to build a different type of provider labor rate model. This was alluded to above but needs further development.

Descriptive vs. Predictive Modeling

The current labor rate model adopts a **descriptive**, rather than a **prescriptive** approach. Because it is trained to model historically accepted rates, it locks us into historical pricing patterns. The implied assumption that this modelling choice makes is that the rates we have paid providers in the past are perfectly optimized. In other words, it assumes that we priced each job at the most competitive rate we could have attained, and based on that assumption it perpetuates past rates onto current jobs. This assumption - that our historical rates are the best rates we could have filled each job at - is not a valid one and is not the approach taken by successful market optimizing architectures like Uber, Amazon Flex, or Airbnb. The danger in this assumption is that in many cases, we actually overpaid providers because we did not have any way to gauge how much lower of a rate the market might have accepted.

This highlights the need for a **prescriptive** model: a model which tells us, not what we paid providers in the past for a similar job, but how likely providers would be to accept a job at any given rate we would like to test. This is what gives us forward-looking predictive capabilities into market behavior, allowing us to proactively scenario-plan and choose minimally viable rates.

Summary of Key Risks in the Current Model's Approach

1. The current model does not offer visibility into the price elasticity curve.
    - The current labor rate model doesn't learn or predict how provider acceptance probability changes across a range of rates.
    - It only knows what has worked historically, then utilizes a costing coefficient to experiment with a slightly lower rate on first posting.
    - This means that you can't run simulations or optimize for margin, because the model doesn't know anything outside its historical window.

        - Even though the model includes features for supply and demand (e.g. provider count, market saturation), it maps these to *historically accepted rates, not to acceptance probability*.
        - That means it doesn't truly capture market dynamics — it only captures how those dynamics happened to correlate with what *we* chose to pay. If we historically overpaid in high-supply markets, the model will learn that high provider count corresponds to high accepted rates - even though the market might have accepted far less.

2. The dynamic costing coefficient, though a clever heuristic tool, is slow and reactive.
    - The dynamic costing coefficient creates a basic adaptive feedback loop, which should allow the system to self-correct over time and move gradually toward minimally viable rates over time. However:
        - It is a very slow and reactive tool. Because it is based on a 6-month rolling average, any market changes take months to register.
        - It's not agile enough to respond to real-time provider sentiment or shifts in labor supply/demand.
        - Contrast: Best-in-class marketplaces like Airbnb or Uber monitor real-time behavior and make dynamic adjustments immediately – often within hours or days, not months.
3. The current model assumes historical rates are baseline optimal.
    - The model's foundation is flawed: it assumes past accepted rates are a valid anchor.
        - The dynamic costing coefficient is a heuristic method to triage this issue, but really it is just taking historical accepted rate as an anchor and giving it a slight haircut to experiment with a lower rate on first posting. It will only offer marginal improvements on rates, but will fail to correct cases of significant overpayment.
4. The current model does not allow for granular targeting.
    - The dynamic costing coefficient applies broadly by region and trade.
    - It doesn't account for provider-specific or hyperlocal behavior (e.g. rural vs urban, weekday vs weekend, emergency vs routine).

    - This leads to **blunt rate adjustments** instead of precision optimization.

5. The current model risks reinforcing legacy pricing inefficiencies.

How To Gather Data for the Proposed Alternative Model

The requisite data to construct our target variable for this model likely already exists within DMG's databases. Data like first posting acceptance/rejection could be used to train a classification model, or time to fill could be used to train a regression or a probabilistic regression (using time to fill as a % of total posting window). We should have enough at this point to build a deployable model, however in order to continually refine the model's output we would need to begin collecting the following kinds of data on provider interactions with the app:

1. Start tracking every provider who sees a rate.
    - Whether they ignore, reject, or accept.
    - Even just tracking "viewed" vs "accepted" would be a big step.
2. Experimentally vary posted rates on certain jobs to generate true A/B data.
3. Log post lifespan - e.g. time until accepted or pulled (if not already tracked)

Exhibit C: The Volume-Feasibility Link

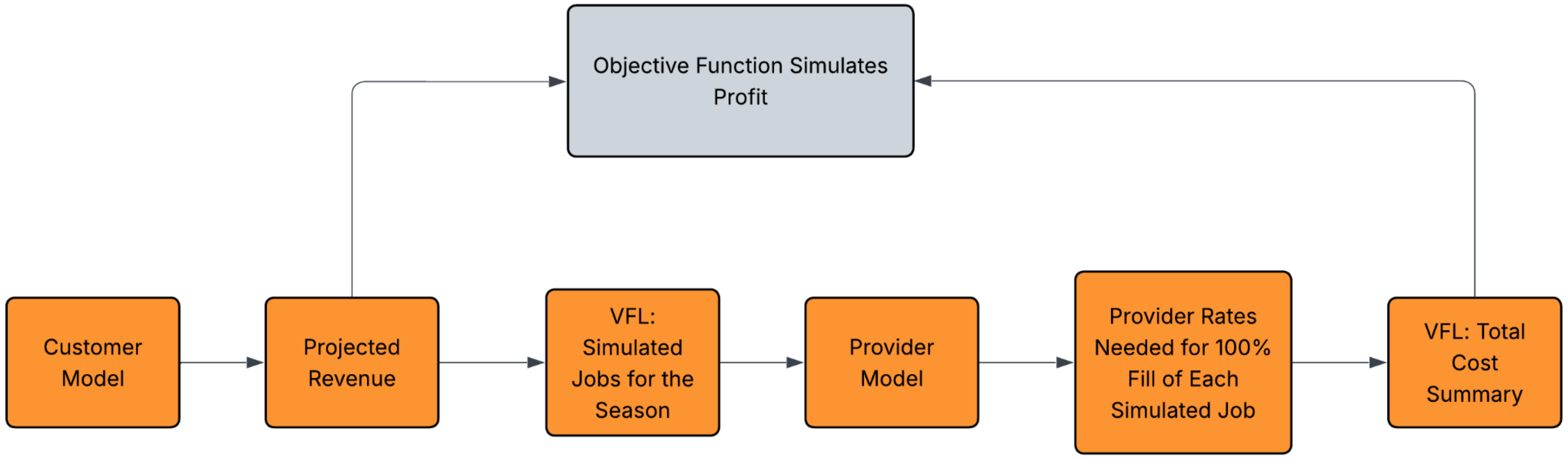
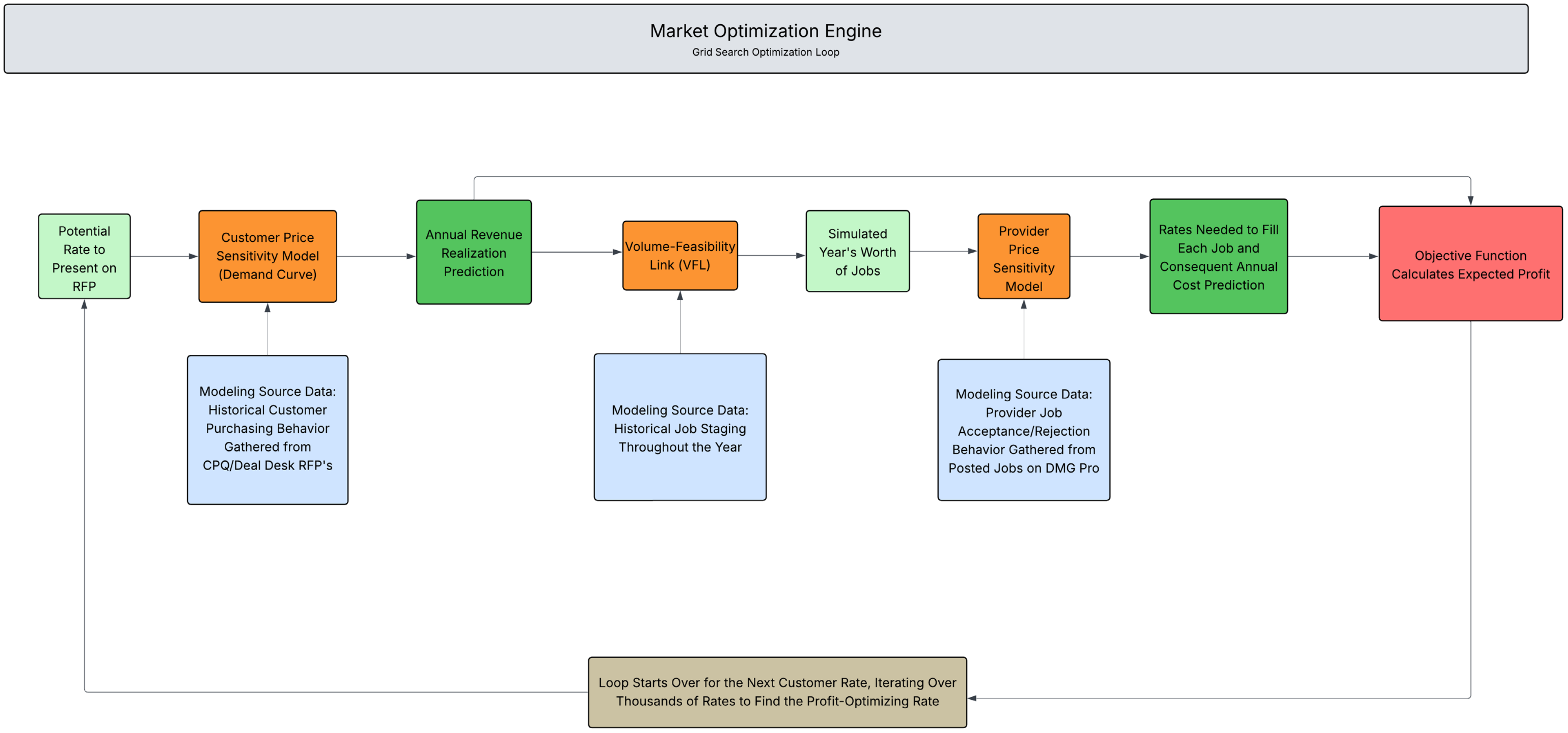

Objective Function Simulates Profit
Customer Model
Projected Revenue
VFL: Simulated Jobs for the Season
Provider Model
Provider Rates Needed for 100% Fill of Each Simulated Job
VFL: Total Cost Summary

Exhibit D: The Complete Optimization Engine


Market Optimization Engine
Grid Search Optimization Loop
Potential Rate to Present on RFP
Customer Price Sensitivity Model (Demand Curve)
Annual Revenue Realization Prediction
Volume-Feasibility Link (VFL)
Simulated Year's Worth of Jobs
Provider Price Sensitivity Model
Rates Needed to Fill Each Job and Consequent Annual Cost Prediction
Objective Function Calculates Expected Profit
Modeling Source Data: Historical Customer Purchasing Behavior Gathered from CPQ/Deal Desk RFP's
Modeling Source Data: Historical Job Staging Throughout the Year
Modeling Source Data: Provider Job Acceptance/Rejection Behavior Gathered from Posted Jobs on DMG Pro
Loop Starts Over for the Next Customer Rate, Iterating Over Thousands of Rates to Find the Profit-Optimizing Rate