Gemini:

This is a very clean, structured visualization of the **Bellman Expectation Equation**, which is a cornerstone of Reinforcement Learning (RL). While the underlying concepts (Markov Decision Processes) are standard, the way this specific document uses **vector notation** to bridge the gap between "expected values" and "matrix operations" is quite effective.
I see this frequently in advanced AI research and control theory, though students often struggle with it. Your document does a great job of breaking down how a single state value v is actually the result of several nested dot products.

Key Takeaways from the Derivation

According to the provided document, the relationship between these components can be broken down into three main levels:

1. The Outcome (o)

The document defines the outcome o as the immediate reward plus the discounted future value:

$o = r + \gamma v'$

This represents a single realization of what you get after taking an action.
It is described as an approximation of the total return g.

2. The State-Action Value (q)

To find the value of taking a specific action a, the document uses a dot product between the probability vector p and the outcome vector o:

$q = p \cdot o = E\,[g|s,a]$
p contains the transition probabilities $p(s'|a,s)$.
o represents the rewards and future values for each possible next state.

3. The State Value (v)

Finally, the value of the current state v is the weighted average of all possible action values, determined by the policy π:

$v = \pi \cdot q = E[g|s]$
π represents the policy $p(a|s)$, or the probability of choosing each action.

What Makes This Unique?

Most textbooks use summation notation ($\sum$). This document leans heavily into **vector geometry**, which is how modern AI frameworks (like PyTorch or TensorFlow) actually compute these values. By treating the policy (π) and transitions (p) as vectors, it turns a complex recursive idea into simple linear algebra.

The diagram on Page 1 perfectly illustrates the "backup" process:

1. Start at next states s'.
2. Weight them by transition dynamics $p1$ to get to the action node $a1$.
3. Weight those by the policy $\pi1$ to "back up" the value to state s.