Structure of the Output Returned by `run_gibbs_cpp()`

The C++ Gibbs sampler returns a list containing three main objects:

- `table_of`: a list of length S (number of saved iterations). Each element is an integer vector of length n, where

  $$\texttt{table\_of[[s]][i]} = \text{table assigned to customer } i$$

  at saved iteration s.

- `dish_of`: a list of length S. Each `dish_of[[s]]` is itself a list of length d (number of views). For each view v, `dish_of[[s]][[v]]` is an integer vector of length T_s, where T_s is the number of tables at iteration s. The entry

  $$\texttt{dish\_of[[s]][[v]][t]} = \text{dish assigned to table } t \text{ in view } v.$$

- `loglik`: a numeric vector of length S containing the log-likelihood at each saved iteration.

This output structure mirrors the hierarchical nature of the model: customers $\to$ tables $\to$ dishes (one per view). Since the number of tables changes over iterations, lists must be used instead of fixed-size arrays.

Example: Interpreting the Structure with a Small Illustration

Assume the following setting:

$$n = 3, \quad d = 2, \quad S = 2.$$

Suppose the saved states returned by the sampler are:

```
res$table_of =
  [[1]]  [1, 1, 2]
  [[2]]  [2, 2, 1]

res$dish_of =
  [[1]]
     [[1]]  [10, 20]
     [[2]]  [11, 21]

  [[2]]
     [[1]]  [30, 40, 50]
     [[2]]  [31, 41, 51]
```

This corresponds to the following interpretation:

State 1 (s = 1).

- Customers:
$$1 \to t_1, \quad 2 \to t_1, \quad 3 \to t_2.$$
- Number of tables: $T_1 = 2$.
- Dishes:
$$\text{View 1: } (10,\ 20)$$
$$\text{View 2: } (11,\ 21)$$
meaning table 1 uses dishes $(10, 11)$, while table 2 uses $(20, 21)$.

State 2 (s = 2).

- Customers:
$$1 \to t_2, \quad 2 \to t_2, \quad 3 \to t_1.$$
- Number of tables: $T_2 = 3$.
- Dishes:
$$\text{View 1: } (30,\ 40,\ 50)$$
$$\text{View 2: } (31,\ 41,\ 51)$$
meaning each table has its own dish in both views.

Why This Structure is Appropriate

- The number of tables T_s varies across iterations; a list-of-lists allows dynamic sizes.
- Accessing the dish used by customer i at iteration s only requires:
$$k_{svi} = \texttt{dish\_of[[s]][[v]][table\_of[[s]][i]]}.$$
- The structure is compact, flexible, and captures the full restaurant–table–dish hierarchy.