Forward-pass complexity (Flat TokenSHAP vs. aHFR-TokenSHAP)

Scope and cost model

We consider attribution for a black-box predictor accessed via a value function $v(S)$, where S is a coalition of active "players" and each evaluation of $v(\cdot)$ corresponds to an expensive forward-pass / model query (e.g., an LLM call). We quantify computational cost by the *number of calls to* $v(\cdot)$ and ignore lower-order CPU overheads (hierarchy sampling, shuffling, bookkeeping).

Notation

- T : number of prompt tokens (players for Flat TokenSHAP).
- $\mathcal{L}$: set of leaf features used in the aHFR estimator; $L = |\mathcal{L}|$.
- $\mathcal{P}$: set of primary-layer nodes with nonempty leaf descendants; $P = |\mathcal{P}|$.
- K : number of Monte Carlo (MC) permutations in the main attribution loop.
- K_0 : number of primary-layer calibration permutations. In the provided implementation, $K_0 = \max(1, \text{round}(0.2K))$.

Flat TokenSHAP: token-level MC Shapley–Shubik

Flat TokenSHAP treats tokens as players and estimates Shapley–Shubik values by sampling K permutations of T tokens. Per permutation, it evaluates $v(\emptyset)$ once and then re-evaluates $v(S)$ after adding each token, giving $(T + 1)$ value-function calls per permutation. Therefore:

$$\#\text{evals}_{\text{Flat}} = K\,(T + 1), \tag{1}$$

which scales as $\Theta(KT)$.

aHFR-TokenSHAP (provided code): leaf-level MC with primary calibration

In the provided `HFR_TokenSHAP.py` implementation under `mixed_players=True` and `adaptive_search=True`, value-function calls arise from two stages:

(1) Primary-layer calibration (group Shapley over $\mathcal{P}$). Calibration runs Monte Carlo group Shapley–Shubik over P primary groups using K_0 permutations. Each calibration permutation evaluates $v(\emptyset)$ once and then once per primary group, for $(P + 1)$ calls:

$$\#\text{evals}_{\text{calib}} = K_0\,(P + 1). \tag{2}$$

(2) Main leaf-level loop (Shapley–Shubik over $\mathcal{L}$). The main estimator accumulates Shapley–Shubik marginals over K sampled *leaf permutations* of length L. Each permutation evaluates $v(\emptyset)$ once and then once per leaf addition, for $(L + 1)$ calls:

$$\#\text{evals}_{\text{main}} = K\,(L + 1). \tag{3}$$

Importantly, in this codepath the main-loop evaluation count depends on L (number of leaves) and does *not* scale with T (prompt length), except insofar as T might influence how features/leaves are defined.

Total. Summing both stages:

$$\#\text{evals}_{\text{aHFR}} = K\,(L + 1) + K_0\,(P + 1), \tag{4}$$

When does aHFR-TokenSHAP outperform Flat TokenSHAP?

Define the (relative) evaluation advantage as the ratio

$$R \stackrel{\text{def}}{=} \frac{\#\text{evals}_{\text{Flat}}}{\#\text{evals}_{\text{aHFR}}} = \frac{K\,(T+1)}{K\,(L+1) + K_0\,(P+1)}. \tag{5}$$

Then aHFR-TokenSHAP uses fewer value-function evaluations than Flat TokenSHAP iff $R > 1$, equivalently:

$$K\,(T+1) \;>\; K\,(L+1) + K_0\,(P+1) \quad\Longleftrightarrow\quad T \;>\; L + \frac{K_0}{K}(P+1). \tag{6}$$

With the default calibration budget $K_0 = \max(1, \text{round}(0.2K))$ and $K \gg 1$ so that $K_0/K \approx 0.2$, the condition becomes

$$T \;\gtrsim\; L + 0.2\,(P+1). \tag{7}$$

Thus, as prompt length T increases while (K, L, P) are held fixed, the Flat TokenSHAP cost grows linearly in T whereas the aHFR-TokenSHAP cost remains approximately constant in T; consequently, aHFR-TokenSHAP *strictly improves* in evaluation-count advantage as T grows, and the reduction percentage approaches 100% as $T \to \infty$.

Percentage reduction metric (Supplementary Fig. S1)

We report the percentage reduction in value-function calls as

$$\text{Reduction}_{\%} = 100\left(1 - \frac{\#\text{evals}_{\text{aHFR}}}{\#\text{evals}_{\text{Flat}}}\right) = 100\left(1 - \frac{K\,(L+1) + K_0\,(P+1)}{K\,(T+1)}\right). \tag{8}$$

Supplementary Fig. S1 visualizes this reduction across a range of (K, T, L, P). Negative values indicate regimes where the calibration + leaf-level procedure exceeds the token-level budget (e.g., when T is small relative to L and the calibration term).

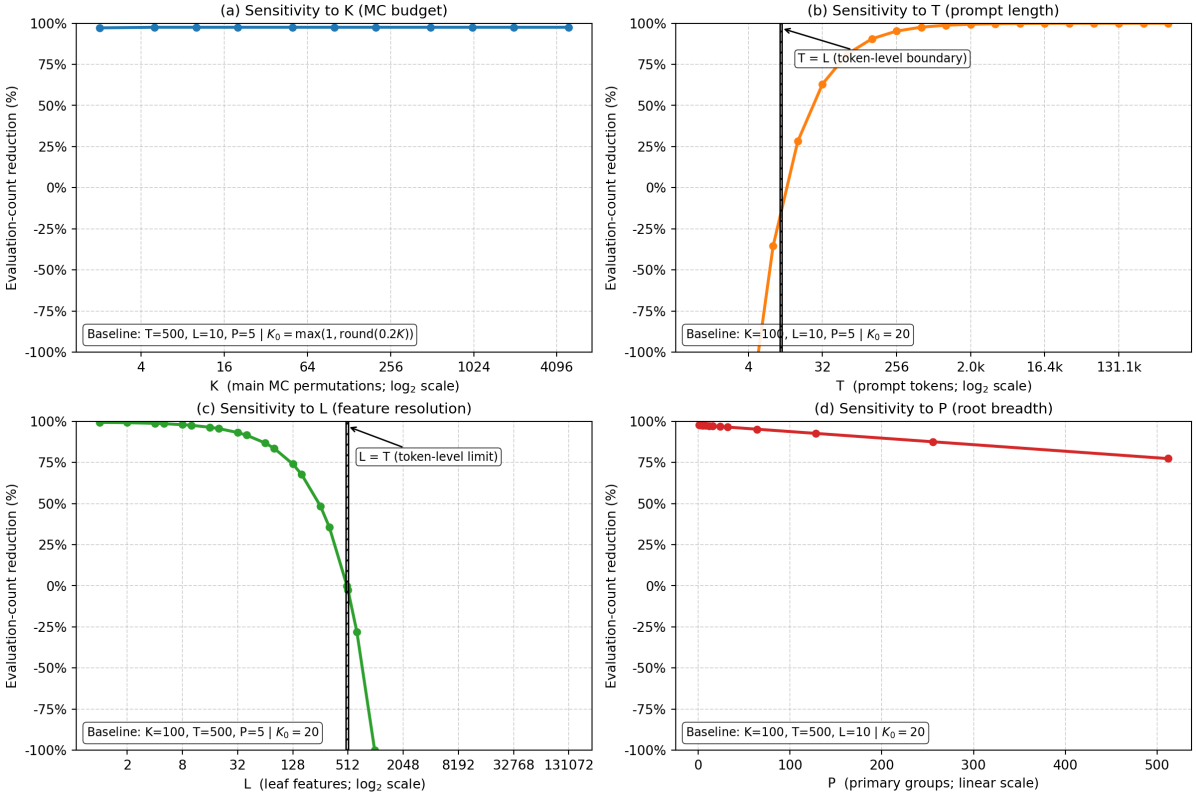
Interpretation caveats

- This analysis compares *counts* of $v(\cdot)$ calls. Wall-clock time may deviate if the per-call token-processing cost varies substantially with S (e.g., different prompt lengths), or if caching/batching is used.
- Flat TokenSHAP and aHFR-TokenSHAP target different explanatory granularities (token vs. leaf-feature). The reduction is relevant when feature/leaf-level attribution is the objective.

Conclusion

Under the provided codepath (`mixed_players=True`, `adaptive_search=True`), the dominant value-function evaluation complexity shifts from $\Theta(KT)$ (Flat TokenSHAP) to $\Theta(KL + K_0P)$ (aHFR-TokenSHAP). Consequently, once prompt length satisfies the threshold in Eq. (6), increasing T strictly improves aHFR-TokenSHAP's evaluation efficiency, yielding substantial forward-pass savings for long prompts.

Figure S1: Percentage reduction in value-function evaluations (aHFR-TokenSHAP vs. Flat TokenSHAP).



Note. The plotted quantity is $100(1 - \#\text{evals}_{\text{aHFR}}/\#\text{evals}_{\text{Flat}})$ with $\#\text{evals}_{\text{Flat}} = K(T+1)$ and $\#\text{evals}_{\text{aHFR}} = K(L+1) + K_0(P+1)$ using $K_0 = \max(1, \text{round}(0.2K))$. Log_2 scaling is used for K, T, and L (linear for P). The baseline in the panels is K=100, T=500, L=10, P=5.