A CRAFT ALGORITHMS

In this section, we present the pseudocode for the three main algorithms in CRAFT, detailing the step-by-step logic of each algorithm. When integrating CRAFT, we leverage vectorized operations to achieve maximum efficiency.

A.1 Estimate Per-Layer Replication Benefit

In CRAFT, the input positive replica counts are chosen from a base-2 geometric progression over $[1, D]$ for computational efficiency (Section 4.2.1).

Algorithm 1 Per-Layer Replication Benefit Estimation

Input 1: list of positive replica counts R
Input 2: number of GPUs D and nodes N
Input 3: offline load distribution W of shape $B \times L \times E$: B - # inference batches, L - # MoE layers, E - # experts per MoE layer, $W[b][\ell][e]$ - token load of expert e at layer ℓ in batch b
Output: map T: keys - replica count $r \in R$; values - length L lists with per-layer GPU balancedness gain compared to placement-only.
Auxiliary Function:
PLACEMENT(W, E, N): returns an expert placement plan P given $L \times E$ load distribution W, expert count E including replicas, and number of nodes N. We use a greedy placement algorithm, similar to EPLB. Placement plan P describes expert-to-device mapping and the number of replicas assigned for each expert.

```
W_s = W.sum(dim = 0)   // W_s shape (L × E)
base = 0.0   // placement-only baseline balancedness
for r in [0] + R do
  placement plan P = PLACEMENT(W_s, E + r, N)
  BAL = [0] * L   // cumulative GPU balancedness
  for b = 0 to B − 1 and ℓ = 0 to L − 1 do
    G = [0] * D   // per-GPU load
    for e = 0 to E − 1 do
      // expert copies include both base and replicas
      r_el = P.num_expert_copies(e, ℓ)
      g_el = P.gpus(e, ℓ)  // GPUs with expert copy
      for g in g_el do
        // per-GPU load
        G[g] += floor(W[b][ℓ][e] / r_el)
      end for
    end for
    BAL[ℓ] += avg(G) / max(G) // balancedness
  end for
  if r == 0 then
    base = BAL[ℓ] / B   // placement-only
  else
    // per-layer average GPU balancedness gain
    T[r][ℓ] = (BAL[ℓ] / B) − base
  end if
end for
return T
```

A.2 Solve Replica Allocation via Dynamic Programming

The time complexity of the following dynamic programming is $O(L \cdot C \cdot |T.keys()|)$, and the space complexity is $O(L \cdot C)$.

Algorithm 2 Replica Allocation Solver Maximizing Replication Benefits

Input 1: map T: replica counts to balancedness gains map, acquired with Algorithm 1
Input 2: expert capacity C (base experts & replicas)
Output: selection list $\mathbf{x}$ of length L with $\mathbf{x}[j] \in 0 \cup \text{keys}(T)$

```
R = T.keys()   // replica counts
L = length(T[R[0]])   // number of MoE layers
// initialize DP states and selections
dp[0..L][0..C] = −∞
dp[0][0] = 0
choice[0..L][0..C] = None
for ℓ = 1 to L do
  j = ℓ − 1
  for c = 0 to C do
    // option 1: no replication for current layer
    dp[ℓ][c] = dp[ℓ − 1][c]
    choice[ℓ][c] = None
    // option 2: iterate through replica counts
    for r in R do
      // check capacity and reachability
      if c ≥ r and dp[ℓ − 1][c − r] > −∞ then
        // calculate new global balancedness gain
        cand = dp[ℓ − 1][c − r] + T[r][j]
        // update DP states if a new best is found
        if cand > dp[ℓ][c] then
          dp[ℓ][c] = cand
          choice[ℓ][c] = r
        end if
      end if
    end for
  end for
end for
x[0..L − 1] = 0   // output initialization
// acquire per-layer replica count from DP states
c = C
for ℓ = L downto 1 do
  r = choice[ℓ][c]
  if r is not None then
    x[ℓ − 1] = r
    c = c − r
  end if
end for
return x
```

A.3 Capacity-Aware Interleaved Expert Assignment

Algorithm 3 Balanced Intra-Layer Interleaved Replica Assignment

Input 1: number of MoE layers L
Input 2: number of GPUs D
Input 3: replica count per-layer $R[0..L-1]$
Output: assignment matrix A of shape $L \times D$, where $A[\ell][g]$ is the number of experts on GPU g for layer ℓ
Auxiliary Functions:
MINCUTOFF$(\mathbf{v}, r)$: returns the r-th smallest value in list $\mathbf{v}$ (1-based).
INTERLEAVESELECT$(\mathcal{I}, k)$: given an ordered list of indices $\mathcal{I}$, returns k indices that are as evenly spaced across $\mathcal{I}$ as possible (including endpoints when $k > 1$).
// assign max number of experts uniformly to all GPUs
// we only need to handle the remainders for each layer
$A[\ell][g] = \left\lfloor \frac{R[\ell]}{D} \right\rfloor$ for $\ell \in [0, L)$ and $g \in [0, D)$
// total # experts per-GPU across all layers, should keep this consistent across all GPUs
$G[g] = \sum_{\ell=0}^{L} A[\ell][g]$ for $g \in [0, D)$
for $\ell = 0$ **to** $L - 1$ **do**
  // only need to handle remainder replicas at each layer
  $r = R[\ell] \bmod D$
  if $r > 0$ **then**
    cutoff = MINCUTOFF(G, r)
    // GPUs with # experts $<$ cutoff are always selected
    $M = [g : g \in G$ and $g <$ cutoff$]$
    $q = r - |M|$
    if $q > 0$ **then**
      // GPUs with tied # experts = cutoff
      $K = sort([g : g \in G$ and $g =$ cutoff$])$
      // select among K in evenly-spread manner
      $J =$ INTERLEAVESELECT(K, q) where $J \subseteq K$
      $S = M \cup J$
    else
      $S = M$
    end if
    // assign replicas to GPUs
    for each $g \in S$ **do**
      $A[\ell][g] += 1$
      G$[g] += 1$
    end for
  end if
end for
return A

B Effect of Replication Factor R

In this section, we evaluate the performance of CRAFT with different replication factors R. Figure 15 depicts the throughput-TTFT curve for `CRA` using various R values under different configurations. We observe a trade-off between R and goodput: When R is too small, the number of replicas is insufficient to fully mitigate load imbalance. When R is too large, excessive replication incurs significant memory overhead and causes slowdowns. This aligns with our observations in Section 5.3. In our deployment setup, $R = 8$ generally achieves the highest goodput across configurations.

C Inter-Token Latency Evaluation

CRAFT maintains the same inter-token-latency (ITL). Figure 12 depicts the decoding ITL in all configurations at their maximum throughputs. We observe that the decoding ITL is stable across different setups. During decoding, batch sizes and token counts are significantly lower than during prefill because decoding requests generate only one or a few tokens (with multi-token prediction) while consuming an increasing amount of KV cache. Thus, load imbalance in MoE blocks becomes less significant, and the attention module constitutes a higher fraction of layer time.

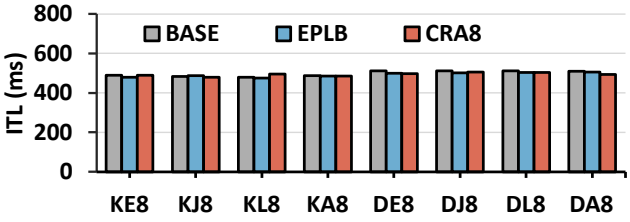
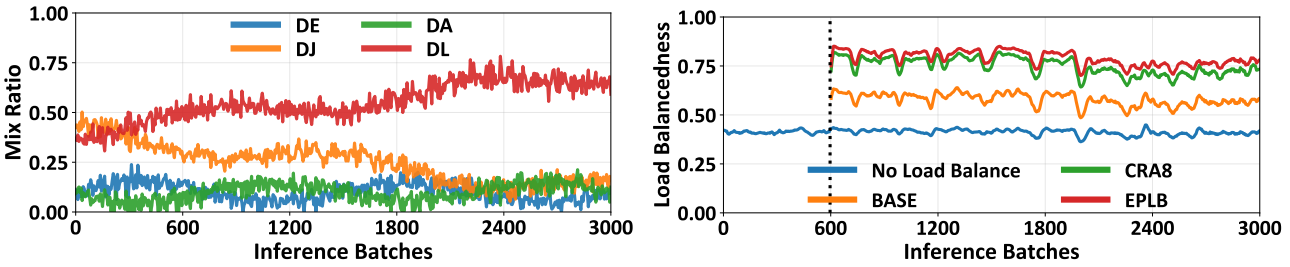


Figure 12. Average inter-token-latency at maximum throughput across different system setups for `BASE`, `EPLB` and `CRA8`.

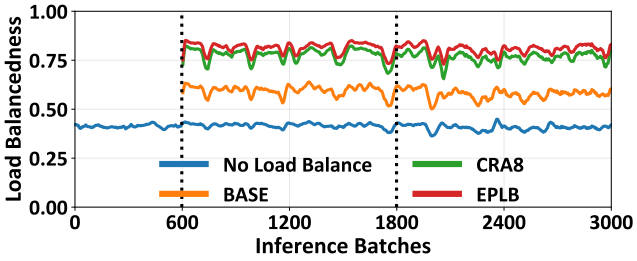
D Impact of Workload Shifts - Additional Workloads

Figures 13 and 14 depict the workload mix ratios and the achieved load balancedness on `SYN2`, `SYN3` and `SYN4`. The experiment setup and observations are identical to Figure 10 in Section 5.6.

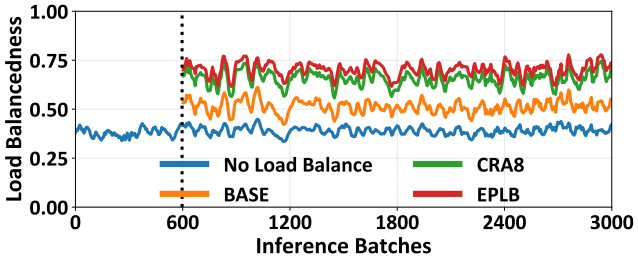


(a) SYN2 workload mix ($\sigma^2 = 0.001$)

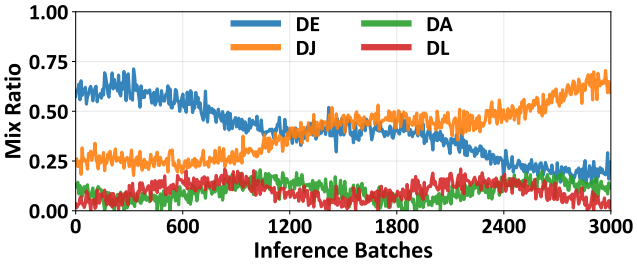
(b) SYN2 balancedness ($\sigma^2 = 0.001$, no rebalancing)



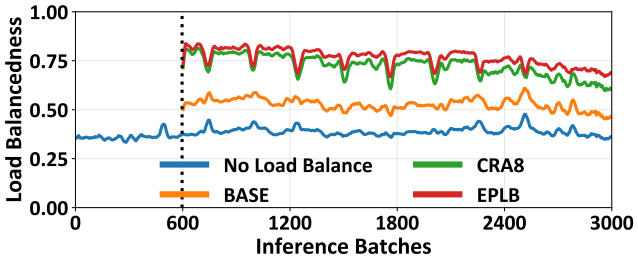
(c) SYN2 balancedness ($\sigma^2 = 0.001$, with rebalancing)



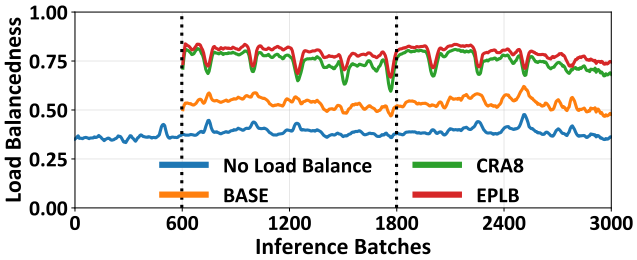
(d) SYN2 balancedness ($\sigma^2 = 0.5$, no rebalancing)



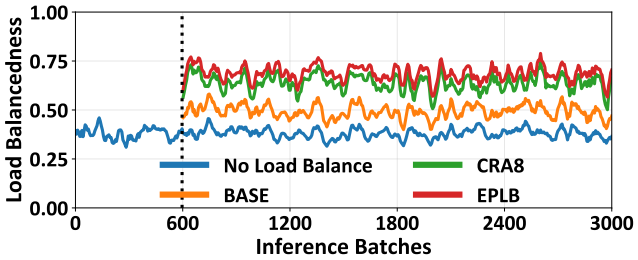
(e) SYN3 workload mix ($\sigma^2 = 0.001$)



(f) SYN3 balancedness ($\sigma^2 = 0.001$, no rebalancing)

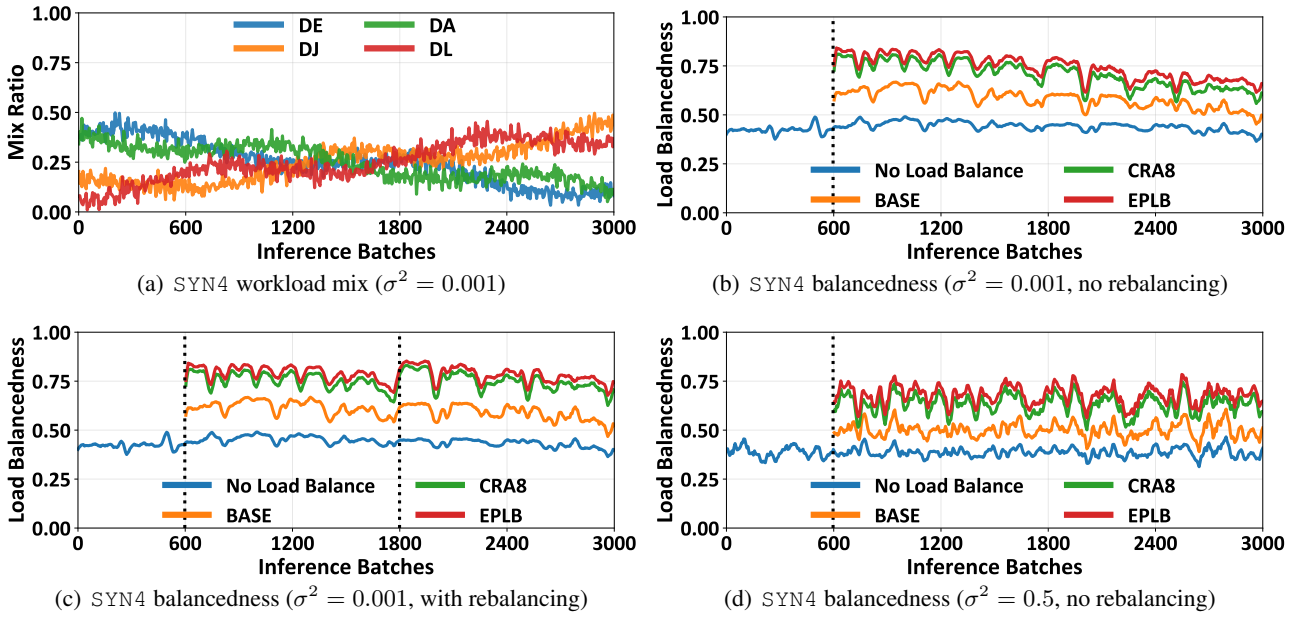


(g) SYN3 balancedness ($\sigma^2 = 0.001$, with rebalancing)



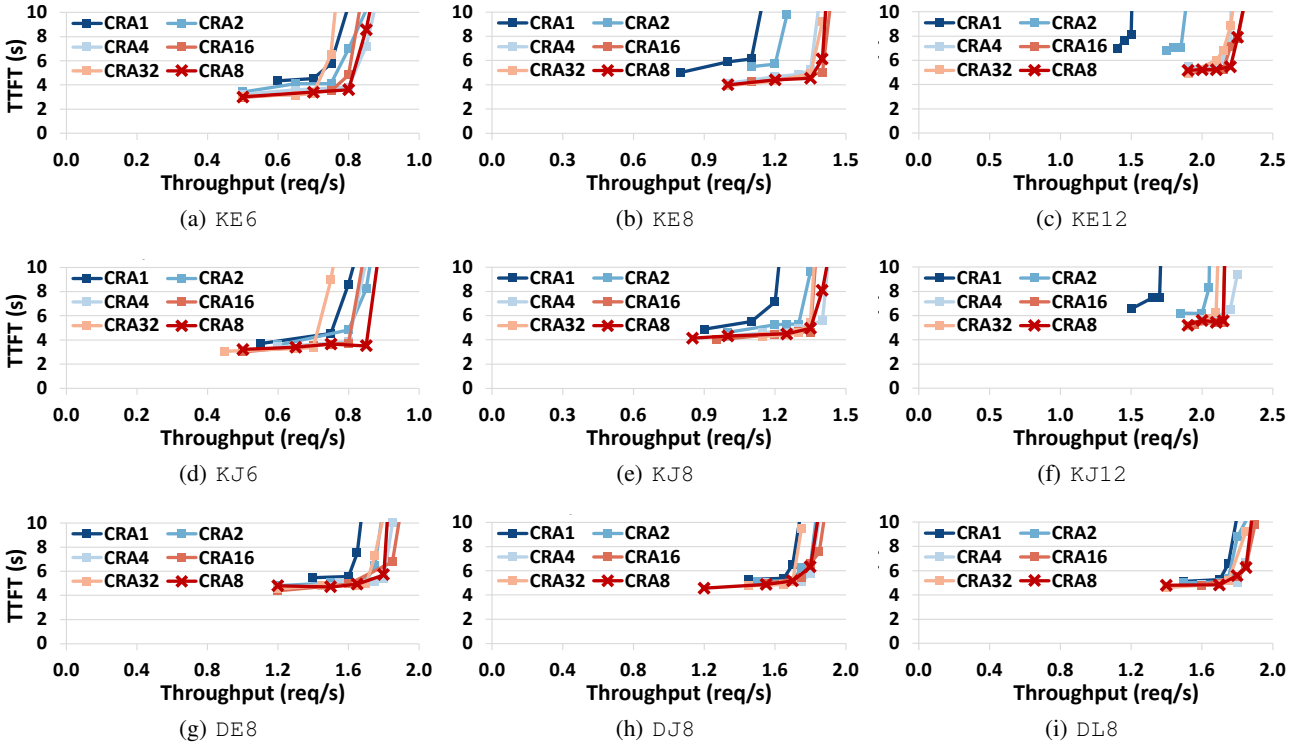
(h) SYN3 balancedness ($\sigma^2 = 0.5$, no rebalancing)

Figure 13. SYN2, SYN3 shifting workload mix ratios and achieved load balancedness over time on DeepSeek-R1-671B (D) across 8 nodes. Figures 13(a) and 13(e) depict the ratio of each dataset over time with burstiness variance $\sigma^2 = 0.001$ for SYN2 and SYN3 respectively. Figures 13(b) - 13(d) and 13(f) - 13(h) depict the achieved load balancedness under various configurations. The dotted lines represent triggered online periodic rebalancing, and data lines are smoothed for clearer presentation. Load-balancing techniques (BASE, EPLB, CRA8) are activated after the initial warmup of 600 iterations. Workloads with higher σ^2 only exhibit higher degree of burstiness; the shift and diurnal patterns remain identical.



(a) SYN4 workload mix ($\sigma^2 = 0.001$)
(b) SYN4 balancedness ($\sigma^2 = 0.001$, no rebalancing)
(c) SYN4 balancedness ($\sigma^2 = 0.001$, with rebalancing)
(d) SYN4 balancedness ($\sigma^2 = 0.5$, no rebalancing)

Figure 14. SYN4 shifting workload mix ratios and achieved load balancedness over time on DeepSeek-R1-671B (D) across 8 nodes. Figure 14(a) depicts the ratio of each dataset over time with burstiness variance $\sigma^2 = 0.001$. Figures 14(b) - 14(d) depict the achieved load balancedness under various configurations. The dotted lines represent triggered online periodic rebalancing, and data lines are smoothed for clearer presentation. Load-balancing techniques (BASE, EPLB, CRA8) are activated after the initial warmup of 600 iterations. Workloads with higher σ^2 only exhibit higher degree of burstiness; the shift and diurnal patterns remain identical.



(a) KE6
(b) KE8
(c) KE12
(d) KJ6
(e) KJ8
(f) KJ12
(g) DE8
(h) DJ8
(i) DL8

Figure 15. End-to-end throughput to TTFT curves on various replication ratios R over different cluster sizes and configurations.

E ARTIFACT APPENDIX

E.1 Abstract

This Artifact Appendix describes how to generate an expert placement and replication plan with CRAFT. Due to the high hardware demands of our evaluations (up to 12 nodes; see Section 3.1), we only provide the core implementation of CRAFT as described in Section 4 and Appendix A. Since the MCKP-based replica allocation algorithm (details in Section 4.2.1 and Appendix A.2) relies on explicit user configuration of the replication factor R, the core implementation instead employs a heuristic-based algorithm that automatically allocates replicas for each layer when the projected per-replica replication benefit is above a relative threshold. This algorithm makes no prior assumption of the memory budget and can be used in real deployment if automatic replica allocation is desired. In practice, we find that this heuristic-based allocation selects a replication factor R close to 8 across all configurations, which is consistent with our observation in Section 5.1 and our evaluation setup (`CRA8`). The provided code references EPLB's official implementation (Chen et al., 2025b). It takes a prior expert load distribution as input and outputs: (1) the number of replicas allocated for each MoE layer by CRAFT, (2) the selected replication ratio R and memory saving relative to EPLB, and (3) the load balancedness of EPLB (placement-only), EPLB, and CRAFT. We show that CRAFT achieves comparable load balancedness while consuming significantly less memory than EPLB.

To facilitate the AE process, we provide links to the GitHub repository and the Zenodo archive that contain the CRAFT source code, expert load distribution traces, and automated workflow scripts to compute and evaluate replication plans. The code runs on Linux, macOS and Windows and is compatible with Intel, Apple Silicon and AMD CPUs. Conda is required. Git-LFS is required when setting up from the GitHub repository.

E.2 Artifact check-list (meta-information)

- **Data set:** custom expert load distribution traces collected during the inference of sampled requests from public datasets FinePDFs, RedPajama-Data-1T and Lambada.
- **Run-time environment:** Requires Linux, macOS or Windows with Conda installations. Git-LFS is required when setting up from the GitHub repository.
- **Hardware:** Intel, Apple Silicon or AMD CPU.
- **Metrics:** replication ratio R and load balancedness.
- **Output:** plain text files containing raw data of all experiments and PNG figures that illustrate the experiment metrics across configurations.
- **Experiments:** CRAFT replication plan generation. Replay-based balancedness computation.
- **How much disk space required (approximately)?:** 2 GB.
- **How much time is needed to prepare workflow (approximately)?:** 5 minutes.
- **How much time is needed to complete experiments (approximately)?:** 15 minutes.
- **Publicly available?:** yes.
- **Code licenses (if publicly available)?:** MIT license.
- **Data licenses (if publicly available)?:** Apache-2.0 license.
- **Archived (provide DOI)?:** 10.5281/zenodo.19022696

E.3 Description

E.3.1 How delivered

Download and extract tarball file `CRAFT_core.tar.gz` from 10.5281/zenodo.19022696 or clone the GitHub repository from https://github.com/Accelsnow/CRAFT_core.

E.3.2 Hardware dependencies

AE should be run on a host machine running Linux, macOS or Windows.

E.3.3 Software dependencies

The verified operating systems and dependencies are listed below:

- Ubuntu 24.04.4 LTS, Windows 11, macOS
- Conda installation
- Git-LFS (when setting up from the GitHub repository)

E.3.4 Data sets

The expert load distribution traces included in the GitHub repository are collected by processing sampled requests from the following three publicly available datasets:

- FinePDFs
- Lambada
- RedPajama-Data-1T

E.4 Installation

After acquiring the code, execute the following commands:

```
$ cd CRAFT_core
$ conda env create --file
    ↪ environment.yml
$ conda activate craft_core
```

If the code is cloned from GitHub without Git-LFS installation, `git clone` will issue a warning about missing `git-lfs`. In this case, continue with conda environment creation and activation. After activating the conda environment, run:

```
git reset --hard
```

E.5 Experiment workflow

To run all experiments and generate the result figure, grant execution permission and run one of the following commands based on the operating system:

Windows:

```
.\run_all.ps1
```

Linux and macOS:

```
./run_all.sh
```

By default, both workflow scripts are configured with a cluster size of 8 nodes. To run a single experiment with custom configurations, view the full list of configuration options by executing:

```
python craft_core.py --help
```

To generate the output figure, place all custom experiment result files under the same directory, then run:

```
python gen_ae_fig.py
    ↪ <custom_directory_path>
```

E.6 Evaluation and expected result

When either `run_all.ps1` or `run_all.sh` finishes successfully, the output files will be stored under the `results` directory in the project root. One `.opt` result file is generated for each input `.pkl` expert load distribution trace file. Eight input trace files are provided by default under the `traces` directory. The naming conventions of the trace and result files follow the configuration syntax described in Section 3.1. The first letter represents the MoE model:

- `D`: DeepSeek-R1-671B
- `K`: Kimi-K2-1000B

The second letter represents the workload:

- `E`: FinePDFs dataset deu_Latn split
- `J`: FinePDFs dataset jpn_Jpan split
- `L`: Lambada dataset
- `A`: RedPajama-Data-1T dataset arxiv split

Each result file contains the following raw data in plain text:

- Number of replicas allocated to each MoE layer with CRAFT.
- The selected CRAFT replication ratio R and the relative memory savings of CRAFT compared to EPLB ↑.
- The load balancedness achieved with EPLB (placement-only), EPLB, and CRAFT ↑.

Our experiment workflow also generates a figure that depicts memory saving and load balancedness across configurations at `results/ae_fig.png`. For experiments with custom configurations, the figure is saved to `<custom_directory_path>/ae_fig.png`.

We expect the results to show that CRAFT consistently achieves comparable load balancedness to EPLB while consuming significantly less memory.