# Supplementary Appendix for: OptiServe

## I. Notations Used

TABLE I
DEFINITION OF THE NOTATIONS USED IN THE MODELLING AND ALGORITHMS.

| Notation | Description |
|---|---|
| $f$ | A serverless function. |
| $G_s$ | Serverless workflow as a graph. |
| $G_{dl}$ | Represents a de-looped graph. |
| $sp$ | A simple path in the workflow. |
| $e = (f_i, f_j)$ | A directed edge from $f_i$ to $f_j$. |
| $\mathcal{P}$ | Power set |
| $ETM(f_i, m)$ | The execution time of general function $f_i$ with memory configuration $m$. |
| $C(f_i, m)$ | The cost of general function $f_i$ with memory configuration $m$. |
| $ETM(f_i, m, a)$ | The execution time of ML inference function $f_i$ with memory configuration $m$ and normalized ML model accuracy $a$ used in the function. |
| $C(f_i, m, a)$ | The cost of ML inference function $f_i$ with memory configuration $m$ and normalized ML model accuracy $a$ used in the function. |
| $M(f_i)$ | Memory usage of function $f_i$. |
| $A(f_i)$ | Normalized accuracy of the ML model used in function $f_i$. |
| $NI(f_i)$ | Number of invocations of function $f_i$. |
| $P(f_i, f_j)$ | Probability of transition from $f_i$ to $f_j$. |
| $EET(G_s)$ | End-to-end execution time of serverless workflow $G_s$. |
| $TP(sp)$ | The transition probability of a simple path. |
| $PET(sp)$ | Execution time of a simple path $s$. |
| $EI(G_c)$ | Expected number of cycle iterations in the cycle structure $G_c$ |
| $EI(G_l)$ | Expected number of loop iterations in the self-loop structure $G_l$ |
| $PGS$ | Price per GB/s execution of functions. |
| $PPI$ | Price per invocation of functions. |
| $EAS(G_s)$ | End-to-end accuracy score of the serverless application $G_s$. |
| $BCR$ | Benefit-cost ratio. |

## II. End-to-End Accuracy Score

Serverless workflows may include multiple ML inference functions, potentially solving heterogeneous tasks (e.g., image classification, object detection, NLP). Consequently, there is no universal, task-agnostic definition of "end-to-end accuracy" that is comparable across all applications. Instead of imposing a one-size-fits-all metric, OptiServe exposes an *application-defined end-to-end accuracy score* (EAS) that acts as a configurable quality proxy for optimization and constraint checking.

*a) Per-function accuracy and normalization.:* For each ML inference function $f \in V$, we assume a finite set of model variants (or configurations) with task-specific evaluation results obtained offline on a fixed validation dataset and metric appropriate for that task (e.g., top-1 accuracy, mAP, F1). We denote the resulting (raw) quality of a chosen variant for $f$ by $\tilde{A}(f)$. Since different functions may use different metrics and ranges, we normalize $\tilde{A}(f)$ to $A(f) \in [0, 1]$ on a per-function basis (e.g., min–max normalization across the variants available for $f$). OptiServe uses only the normalized values $A(f)$ in the optimization.

*b) Accuracy aggregation as an application interface.:* Let $\mathcal{F}_{ML} \subseteq V$ be the set of ML inference functions in the workflow. We define the end-to-end accuracy score as

$$EAS(G_s) \triangleq \Phi\big(\{A(f)\}_{f \in \mathcal{F}_{ML}}\big), \tag{1}$$

where $\Phi(\cdot)$ is a user-provided aggregation function that reflects application semantics (e.g., which stages are more important, whether errors compound, etc.). OptiServe requires $\Phi$ to satisfy the following *contract*:

- **Monotonicity:** $EAS(G_s)$ is non-decreasing in each $A(f)$. Upgrading a model variant for any ML function cannot reduce the workflow score.
- **Boundedness/Normalization:** $EAS(G_s)$ must be mapped to a bounded range (e.g., $[0, 1]$) to enable meaningful constraints such as $EAS(G_s) \geq EASC$.
- **Task consistency:** each $A(f)$ is computed using a fixed validation set and metric per task to ensure comparable variant ordering within the same function.

These requirements are sufficient for feasibility checks and for the greedy heuristics used by OptiServe, while avoiding assumptions about cross-task error correlations that are application-specific.

*c) Recommended templates.:* OptiServe supports any $\Phi$ satisfying the above contract. In practice, we recommend the following common templates:

$$\textbf{Weighted sum: } EAS(G_s) = \sum_{f \in \mathcal{F}_{ML}} w_f \cdot A(f), \quad \sum_f w_f = 1,\ w_f \geq 0; \tag{2}$$

$$\textbf{Bottleneck (min): } EAS(G_s) = \min_{f \in \mathcal{F}_{ML}} A(f); \tag{3}$$

$$\textbf{Geometric mean: } EAS(G_s) = \prod_{f \in \mathcal{F}_{ML}} A(f)^{w_f}, \quad \sum_f w_f = 1,\ w_f \geq 0. \tag{4}$$

The weighted-sum template is useful when stages have different importance, the bottleneck template models pipelines dominated by the weakest stage, and the geometric mean captures compounding correctness across stages.

*d) Example.:* For a workflow with ML functions $f_1$–$f_4$, an instance of the weighted-sum template is:

$$EAS(G_s) = w_1 A(f_1) + w_2 A(f_2) + w_3 A(f_3) + w_4 A(f_4), \tag{5}$$

where weights $\{w_i\}$ encode application priorities (e.g., upstream gating stages may be assigned larger weights). The accuracy constraint $EAS(G_s) \geq EASC$ thus enforces a minimum acceptable quality level under the selected proxy.

## III. Performance and Cost Modelling of some Functions

Figure 1 presents the performance and cost models for several real-world functions, along with their modelling accuracy.

## IV. Possible Structures of Serverless Workflows

To understand the modelling process, it is essential first to analyze the structural patterns that can appear within a workflow graph. As illustrated in Figure 2, serverless workflows commonly exhibit four fundamental structures: parallelism, branching, cycles, and self-loops. The process of simplifying the application workflow for performance and cost modelling—along with the handling of each structural pattern—is detailed in section V and section VI.

## V. Application Performance Modelling

We propose a performance model to estimate the end-to-end execution time of a serverless application. To do this, we transform the original serverless workflow $G_s$ into a probabilistic directed acyclic graph (DAG), denoted as $G_{ET}$. The execution time model is defined as follows:

$$\mathrm{G}_{perf} = (V, E, P, M, A, ET, ETTP) \tag{6}$$

- In which $G_{perf}$ is a DAG without any loops and cycles;

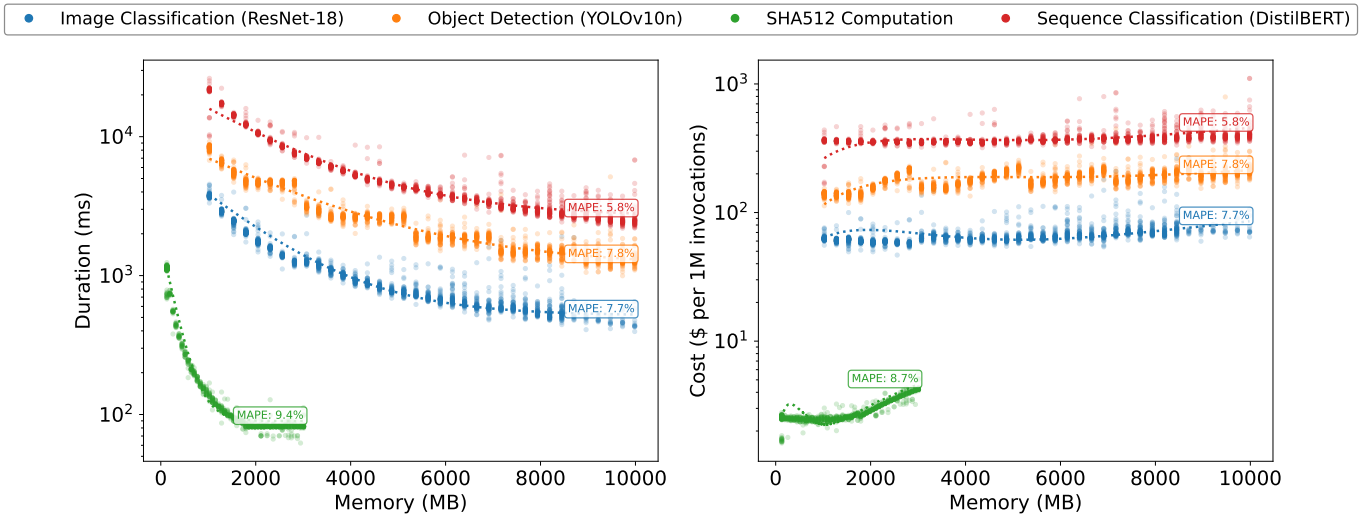


Fig. 1. These figures show the actual execution times of several functions alongside their model predictions. Each function was invoked 100 times per memory configuration, with the resulting durations plotted as translucent scatter points. The execution time and cost models appear as dotted lines, and the $MAPE$ values—indicating the prediction error relative to the observed medians—are annotated near each curve.

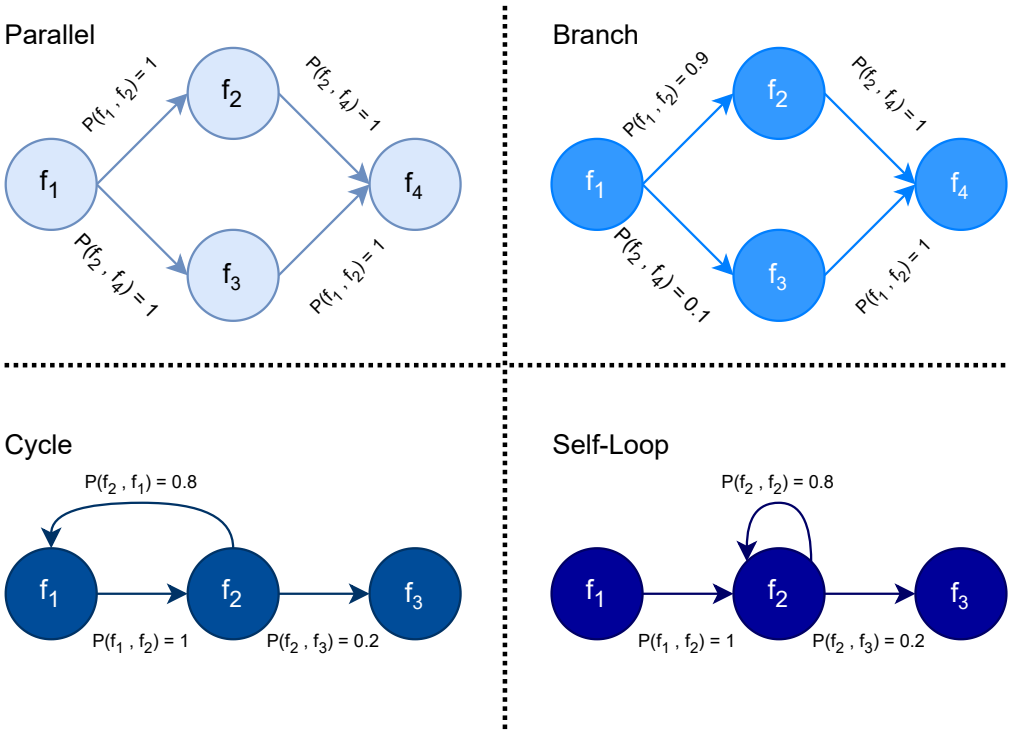


Fig. 2. All the possible structures within a workflow. Adopted from [1].

- $\sum_{s \in ASP(f_{start}, f_{end})} TP(sp) = 1$, this ensures that the transition probabilities of all simple paths from the start node to the end node in $G_{perf}$ add up to 1.
- $\mathbf{P} : \mathbf{V} \times \mathbf{V} \to [0, 1]$ is a transition probability function. $P(f_i, f_j)$ stands for the probability of invoking $f_j$ after $f_i$ is executed.

Having these conditions, the end-to-end execution time of the serverless workflow $G_s$, denoted as $EET(G_s)$, is calculated as follows:

$$EET(G_s) = \sum_{s \in ASP(f_{start}, f_{end})} TP(sp) \cdot PET(sp) \tag{7}$$

where $ASP(f_{start}, f_{end})$ denotes the set of all simple paths between the start and end nodes in $G_{perf}$, which is the probabilistic DAG derived from the original workflow $G_s$ using our execution time model. To construct this probabilistic DAG, the model eliminates cycles and self-loops and prunes parallel paths. The process involves handling the four fundamental workflow structures, which we describe in the following subsections.

**Branches.** In a branch structure, the workflow follows only one of the available paths based on the condition that holds at runtime. Each condition is associated with a probability, defined by the transition probability function. To simplify such a structure, we introduce a temporary node, $f_B$, that replaces the entire branch. The response time of $f_B$ is set to the weighted average of the response times of all branch paths. Importantly, we also preserve the transition probabilities of each path, as they are needed later in the $ETTP$ computation.

**Parallels.** In a parallel structure between $f_i$ and $f_j$, the workflow executes all parallel paths following $f_i$, and proceeds to $f_j$ only after all paths complete. Based on this behaviour, we replace the parallel block with a single node $f_P$, whose execution time is set to the longest execution time among the parallel paths. If any of the paths contain $f_B$ nodes, we also update the $ETTP$ list accordingly.

**Cycles.** To process cycles, we replace each cycle structure $G_c$ (starting at $f_i$ and ending at $f_j$) with a single node. The execution time of this node accounts for the expected number of times the cycle is executed. We define this expected iteration count as $EI(G_c)$, which represents the average number of times the workflow re-enters the cycle after reaching $f_j$. It can be computed as:

$$\begin{aligned} EI(G_c) &= \sum_{n=1}^{\infty} \big[1 - P(f_i, f_j)\big] \cdot \big[P(f_i, f_j)\big]^{n-1} \cdot (n-1) \\ &= \frac{P(f_i, f_j)}{1 - P(f_i, f_j)} \end{aligned} \tag{8}$$

The expected execution time of the cycle is then calculated by multiplying this average iteration count by the time required for one cycle iteration. The total execution time of the node that replaces the cycle is the sum of the execution time of $f_i$ and this expected value.

**Self-Loops.** Similar to the approach used for processing cycles, we handle a self-loop structure $G_l$ consisting of only $f_i$ by estimating the expected number of loop iterations. This is given by:

$$EI(G_l) = \frac{P(f_i, f_i)}{1 - P(f_i, f_i)} \tag{9}$$

The expected execution time of the self-loop is then calculated by summing the execution time of $f_i$, denoted as $ET(f_i)$, and the product of the expected number of iterations and $ET(f_i)$. This gives:

$$ET(G_l) = [1 + EI(G_l)] \cdot ET(f_i)$$

This transformation allows us to replace the self-loop structure with a single node that captures its expected execution cost. Figure 3 illustrates an example of how an initial serverless workflow is transformed into a probabilistic DAG using the processing techniques described earlier.

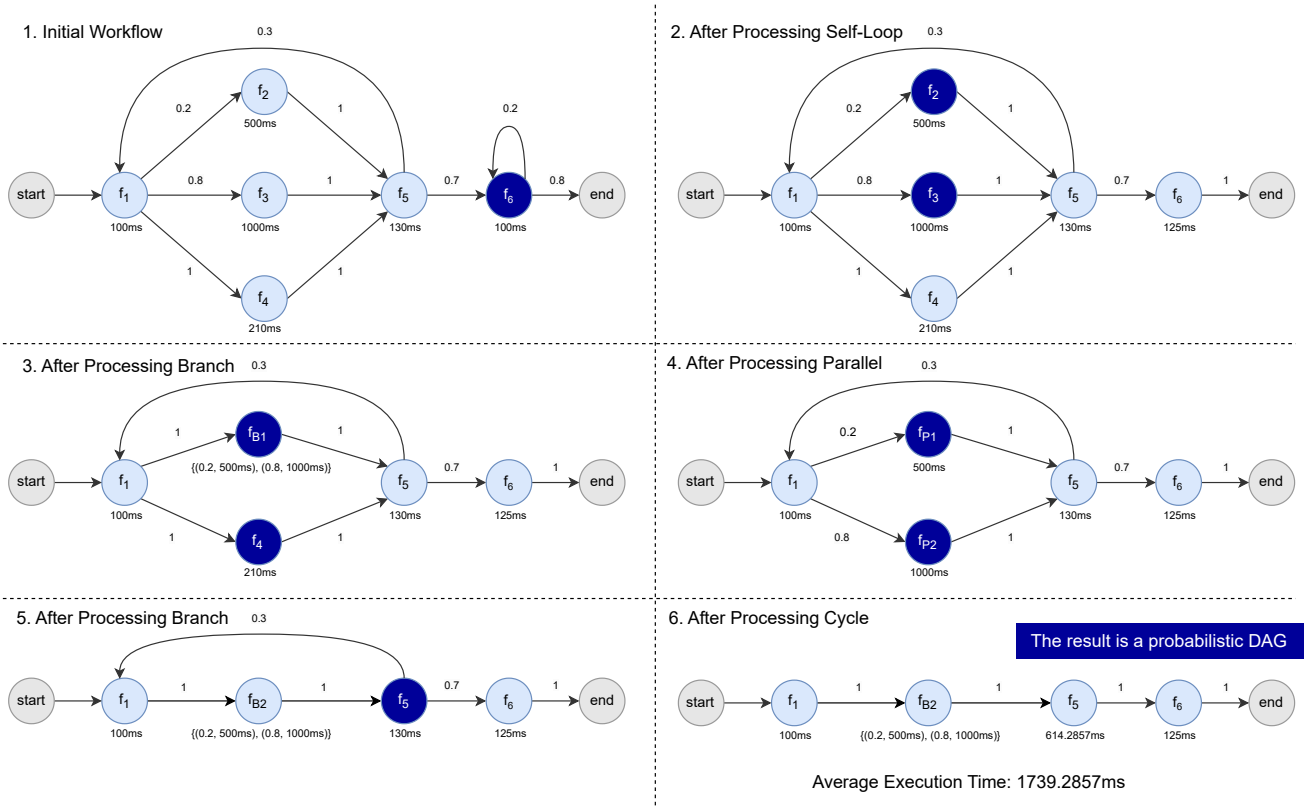


Fig. 3. An illustration of the performance modelling process for a serverless workflow. The numbers on the edges represent transition probabilities between functions. For each $f_B$ node, $ETTP$ is shown as a set of (probability, execution time) tuples. After converting the workflow into a DAG, we compute the average execution time by aggregating the execution times of all nodes, weighted by their $ETTP$ values.

## VI. Application Cost Modelling

We define the following cost model to estimate the average cost of a serverless workflow. Under the standard GB-second pricing model, the cost of each function depends on three main factors: the allocated memory size, the rounded-up execution time (which varies based on both the memory size and the ML model used for inference), and the number of times the function is invoked. Taking these factors into account, we express the cost model as follows:

$$G_{cost} = (V, E, P, M, A, ET, NI, C) \tag{10}$$

The cost modelling process begins by eliminating cycles and self-loops, as outlined in section V. Once the workflow has been de-looped, we account for the effects of parallel and branching structures. Specifically, the number of invocations $NI$ for each node is updated based on the sum of transition probabilities across all simple paths from the start node to that node. After updating the invocation counts, we compute the cost of each function—both general-purpose and ML inference functions—using. The total workflow cost is then obtained by summing the costs of all remaining functions.

## VII. Computational Complexity of the Optimizations

We solve three separate optimization problems: (i) minimize end-to-end execution time given budget and accuracy constraints, (ii) minimize cost given performance and accuracy constraints , (iii) maximize accuracy given performance and budget constraints . Each problem has one objective and exactly two

constraints. We prove that all of these problems are NP-hard by showing that their *decision* version is NP-complete.

*a) Model and notation.:* For each function $f \in V$ in the workflow $G_s = (V, E)$, let $\Pi_f$ be the finite set of memory options and $\Delta_f$ the finite set of model variants. A choice $(\pi_f, \delta_f) \in \Pi_f \times \Delta_f$ induces cost $C_f(\pi_f, \delta_f)$, execution time $ET_f(\pi_f, \delta_f)$, and accuracy contribution $\delta_f$. Given a full assignment $(\pi, \delta)$, we can compute

$$C^{\pi,\delta}(G_s), \quad EET^{\pi,\delta}(G_s), \quad EAS^{\delta}(G_s)$$

in polynomial time (a single traversal over $G_s$ suffices).

*b) Decision versions:* For each optimization goal, we define a decision problem by bounding the objective with a threshold.

**PERF-DEC:** Given bounds $BC$, $EASC$, and $P$, does there exist $(\pi, \delta)$ with $\sum_{i=1}^{n} C^{\pi,\delta}(f_i) \leq BC$, $EAS^{\delta}(G_s) \geq EASC$, and $EET^{\pi,\delta}(G_s) \leq P$?

**COST-DEC:** Given bounds $PC$, $EASC$, and $B$, does there exist $(\pi, \delta)$ with $EET^{\pi,\delta}(G_s) \leq PC$, $EAS^{\delta}(G_s) \geq EASC$, and $\sum_{i=1}^{n} C^{\pi,\delta}(f_i) \leq B$?

**ACC-DEC:** Given bounds $PC$, $BC$, and $A$, does there exist $(\pi, \delta)$ with $EET^{\pi,\delta}(G_s) \leq PC$, $\sum_{i=1}^{n} C^{\pi,\delta}(f_i) \leq BC$, and $EAS^{\delta}(G_s) \geq A$?

**Theorem 1.** PERF-DEC, COST-DEC, *and* ACC-DEC *are NP-complete. Therefore, the optimization problems in the paper are NP-hard.*

*Proof. In NP.* For any $(\pi, \delta)$ we can evaluate $C^{\pi,\delta}(G_s)$, $EET^{\pi,\delta}(G_s)$, and $EAS^{\delta}(G_s)$ in polynomial time, so all three decision problems can be verified in polynomial time.

*NP-hardness.* We reduce from the 0/1 *Multi-dimensional Knapsack Problem* (MDKP) with two resource constraints, which is NP-complete [2], [3]. An MDKP instance gives $n$ items, each with value $v_i$ and two weights $w_i^{(1)}$ and $w_i^{(2)}$, capacities $W^{(1)}, W^{(2)}$, and a required total value $V$. The question is whether there exists a subset $S$ such that

$$\sum_{i \in S} w_i^{(1)} \leq W^{(1)}, \quad \sum_{i \in S} w_i^{(2)} \leq W^{(2)}, \quad \sum_{i \in S} v_i \geq V.$$

Construct a workflow $G_s$ that is a chain of $n$ functions $f_1, \ldots, f_n$. For each $f_i$ create exactly two configuration pairs:

$$(\pi_i^0, \delta_i^0) \text{ (skip)} \quad \text{and} \quad (\pi_i^1, \delta_i^1) \text{ (take)}.$$

Define

$$C^{\pi_i^0,\delta_i^0}(f_i) = 0, \quad ET^{\pi_i^0,\delta_i^0}(f_i) = 0, \quad \delta_i^0(f_i) = 0,$$

$$C^{\pi_i^1,\delta_i^1}(f_i) = w_i^{(1)}, \quad ET^{\pi_i^1,\delta_i^1}(f_i) = w_i^{(2)}, \quad \delta_i^1(f_i) = v_i.$$

Set the global thresholds as follows and observe which decision problem we are answering:
*For* PERF-DEC*:* set $BC = W^{(1)}$, $EASC = V$, and $P = W^{(2)}$.
*For* COST-DEC*:* set $PC = W^{(2)}$, $EASC = V$, and $B = W^{(1)}$.
*For* ACC-DEC*:* set $PC = W^{(2)}$, $BC = W^{(1)}$, and $A = V$.

Selecting $(\pi_i^1, \delta_i^1)$ corresponds to including item $i$ in the knapsack. In our system, the end-to-end accuracy $EAS^{\delta}(G_s)$ is computed by a formula. For the hardness proof, we intentionally restrict to a special case in which this formula collapses to a plain sum of per-function accuracies. Since the workflow DAG is a chain and metrics are additive, the three inequalities in each decision problem exactly match the MDKP constraints and value requirement. Hence, a feasible configuration exists if and only if the MDKP instance is feasible. The reduction is polynomial.

Thus, each decision problem is NP-hard and, with membership in NP, NP-complete. Any polynomial-time optimal solver for the corresponding optimization problem would decide its decision version, so each optimization problem is NP-hard. □

*c) Remark on generality.:* The reduction employs a highly restricted instance: a linear DAG, two options per node, additive aggregation, and accuracy that depends solely on the model. Our real setting allows for arbitrary DAGs, numerous options, and non-linear path effects, so the full problems are at least as challenging.

*d) Implication.:* Since the decision versions of our three configuration problems are NP-complete, the corresponding optimization problems are NP-hard. Thus, computing exact optimal solutions is impractical for realistic workflows with dozens of memory and model options. We therefore turn to heuristic search strategies that trade optimality guarantees for tractable runtime. Our design goals are (i) fast convergence, (ii) good solution quality in practice, and (iii) scalability with respect to the number of functions and configuration choices.

# VIII. Tables and Algorithms

TABLE II
UNIFIED SUMMARY OF FUNCTION BENCHMARKS AND CORRESPONDING ML MODEL VARIANTS WITH PARAMETER COUNTS IN MILLIONS.

| Function | Type | Description | Model Variant | Param (M) |
|---|---|---|---|---|
| FileWriteDelete | General | disk I/O-intensive | NA | NA |
| HashCompute | General | CPU-intensive | NA | NA |
| NetworkDownload | General | Network-intensive | NA | NA |
| Image Classification | ML Inference | Using ResNet variants. | ResNet-18 [4]<br>ResNet-34<br>ResNet-50<br>ResNet-101<br>ResNet-152 | 11.7<br>21.8<br>25.6<br>44.5<br>60.2 |
| Object Detection | ML Inference | Using YOLO variants. | YOLOv10n [5]<br>YOLOv10s<br>YOLOv10m<br>YOLOv10l | ∼2.0<br>∼11.0<br>∼25.0<br>∼49.0 |
| Sequence Classification | ML Inference | Using BERT variants. | DistilBERT-base-uncased [6]<br>BERT-base-uncased [7]<br>RoBERTa-base [8] | 66<br>110<br>125 |

For the image classification and object detection tasks, we use a batch of 24 images during evaluation. For sequence classification, we evaluate the model on a batch of 16 text samples, each composed of 8 distinct paragraphs.

Notably, the general functions were deployed directly on AWS Lambda using the standard ZIP method. For the ML functions, we used container image deployment to gain full control over the runtime environment and dependencies, which also made it easier to include custom libraries.

# References


[1] C. Lin and H. Khazaei, "Modeling and optimization of performance and cost of serverless applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 615–632, 2020.

[2] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[6] V. Sanh, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter." in *Proceedings of Thirty-third Conference on Neural Information Processing Systems (NIPS2019)*, 2019.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, 2019, pp. 4171–4186.

[8] L. Zhuang, L. Wayne, S. Ya, and Z. Jun, "A robustly optimized BERT pre-training approach with post-training," in *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, S. Li, M. Sun, Y. Liu, H. Wu, K. Liu, W. Che, S. He, and G. Rao, Eds., Huhhot, China, Aug. 2021, pp. 1218–1227. [Online]. Available: https://aclanthology.org/2021.ccl-1.108/

**Algorithm 1:** MinimumFeasibleModelConfig

**Input :** Budget constraint $BC$, accuracy constraint $EASC$, serverless workflow $G_s = (V, E, P, M, A, ET, ETTP, NI, C, MLIST, ALIST, ETM)$

**Output:** Memory configuration $\pi$ and model configuration $\delta$ satisfying the constraints

$A \leftarrow A[f \mapsto \min(\text{ALIST}(f))], \forall f \in V$ ; // Use minimal model configuration $\delta_{min}$

$G_{dl} \leftarrow (V, E', P', M, A, ET, NI, C, MLIST, ALIST, ETM)$ ; // De-looped graph

$NSP \leftarrow |\text{ASP}_{G_{dl}}(f_{str}, f_{end})|$ ; // Number of simple paths in $G_{dl}$

$\pi_{curr} \leftarrow \pi_{min}$ ; // Initial memory configuration

$\delta_{curr} \leftarrow \delta_{min}$ ; // Initial model configuration

$C_{curr} \leftarrow \sum_{i=1}^{n} C^{\pi_{min}, \delta_{min}}(f_i)$ ; // Current cost

$EAS_{curr} \leftarrow EAS^{\delta_{min}}(G_s)$ ; // Current end-to-end accuracy score

$t \leftarrow 1$;

**while** *BC* - $C_{curr} > 0$ *and* $EAS_{curr} < EASC$ *and* $t \leq NSP$ **do**

  $s_{cp} \leftarrow \text{FindCriticalPath}(G_{dl}, t)$ ; // t-th critical path

  **foreach** $f_i \in s_{cp}$ **do**

    **foreach** $acc_j \in$ *ALIST*$(f_i)$ **do**

      Compute $\Delta EAS(f_i, acc_j)$ and $\Delta C(f_i, \pi_{curr}(f_i), acc_j)$;

  $(f_{tmp}, acc_{tmp}) \leftarrow \arg\max_{f_u, acc_v} BCR(f_u, acc_v)$

    s.t. $\Delta C(f_u, \pi_{curr}(f_i), acc_v) \leq BC - C_{curr}$ ;

  In cases where multiple model configurations yield the same maximum $BCR$, we break the tie by choosing the one that results in the smallest increase in cost.

  **if** $f_{tmp}$ *and* $acc_{tmp}$ *exist* **then**

    $A[f_{tmp}] \leftarrow acc_{tmp}$;

    $ET[f_{tmp}] \leftarrow ETM(f_{tmp}, \pi_{curr}(f_{tmp}), acc_{tmp})$;

    $\delta_{curr}(f_{tmp}) \leftarrow acc_{tmp}$;

    $C_{curr} \leftarrow C_{curr} + \Delta C(f_{tmp}, \pi_{curr}(f_{tmp}), acc_{tmp})$;

    $EAS_{curr} \leftarrow EAS^{\delta_{curr}}(G_s)$;

  **else**

    $t \leftarrow t + 1$;

**return** $\delta_{curr}$

**Algorithm 2:** Performance Optimization with Budget and Accuracy Constraints

**Input :** Budget constraint $BC$, accuracy constraint $EASC$, serverless workflow $G_s = (V, E, P, M, A, ET, ETTP, NI, C, MLIST, ALIST, ETM)$

**Output:** Memory configuration $\pi$ and model configuration $\delta$ satisfying the constraints

$M \leftarrow M[f \mapsto \min(\text{MLIST}(f))], \forall f \in V$ ; // Use minimal memory configuration $\pi_{min}$

$G_{dl} \leftarrow (V, E', P', M, A, ET, NI, C, MLIST, ALIST, ETM)$ ; // De-looped graph

$NSP \leftarrow |\text{ASP}_{G_{dl}}(f_{str}, f_{end})|$ ; // Number of simple paths in $G_{dl}$

$\pi_{curr} \leftarrow \pi_{min}$ ; // Initial memory configuration

$\delta_{curr} \leftarrow \text{MinimumFeasibleModelConfig}(BC, EAS, G_s)$ ; // Minimal feasible model configuration

$C_{curr} \leftarrow \sum_{i=1}^{n} C^{\pi_{min}, \delta_{curr}}(f_i)$ ; // Current cost

$EET_{curr} \leftarrow EET^{\pi_{min}, \delta_{curr}}(G_s)$ ; // Initial end-to-end runtime

$t \leftarrow 1$;

**while** $BC - C_{curr} > 0$ *and* $t \leq NSP$ **do**

  $s_{cp} \leftarrow \text{FindCriticalPath}(G_{dl}, t)$ ; // t-th critical path

  **foreach** $f_i \in s_{cp}$ **do**

    **foreach** $mem_j \in \textit{MLIST}(f_i)$ **do**

      Compute $\Delta EET(f_i, mem_j, \delta_{curr}(f_i))$ and $\Delta C(f_i, mem_j, \delta_{curr}(f_i))$;

  **if** $\exists mem_v \in \textit{MLIST}(f_u) : \Delta EET(f_u, mem_v, \delta_{curr}(f_u)) \leq 0 \wedge \Delta C(f_u, mem_v, \delta_{curr}(f_u)) < 0$ **then**

    $(f_{tmp}, mem_{tmp}) \leftarrow \arg\min_{f_u, mem_v} \Delta C(f_u, mem_v, \delta_{curr}(f_u))$

  **else**

    $(f_{tmp}, mem_{tmp}) \leftarrow \arg\min_{f_u, mem_v} \Delta EET(f_u, mem_v, \delta_{curr}(f_u))$

      s.t. $\Delta C(f_u, mem_v, \delta_{curr}(f_u)) \leq BC - C_{curr}$ and $\Delta EET(f_u, mem_v, \delta_{curr}(f_u)) < 0$;

    This is the stage where both $BCR_{\text{max}}$ and $BCR_{\text{slope}}$ are applied. We rank options using their corresponding $BCR$ values. In cases where multiple memory configurations yield the same maximum $BCR$, we break the tie by choosing the one that results in the smallest increase in cost.

  **if** $f_{tmp}$ *and* $mem_{tmp}$ *exist* **then**

    $M[f_{tmp}] \leftarrow mem_{tmp}$;

    $ET[f_{tmp}] \leftarrow ETM(f_{tmp}, mem_{tmp}, \delta_{curr}(f_{tmp}))$;

    $\pi_{curr}(f_{tmp}) \leftarrow mem_{tmp}$;

    $C_{curr} \leftarrow C_{curr} + \Delta C(f_{tmp}, mem_{tmp}, \delta_{curr}(f_{tmp}))$;

    $EET_{curr} \leftarrow EET_{curr} + \Delta EET(f_{tmp}, mem_{tmp}, \delta_{curr}(f_{tmp}))$;

  **else**

    $t \leftarrow t + 1$;

**return** $\pi_{curr}, \delta_{curr}$