

his supplementary information reports additional robustness, ablation, sensitivity, exact-optimization, resource-utilization, budget-sweep, and reproducibility results for BudgetFunnel.

Supplementary Information for BudgetFunnel: Coarse-to-Fine Global Budget Allocation for Trace-Driven Serverless Cold-Start Mitigation

Zhipeng Ye³, Zihao Lu⁴, Yuan Zhang⁴, Jiayi Hong⁴,
Zhibin Shao⁵, Xuming Wu^{1*}, Gaoyuan Zhao^{2*}, Yan Cao⁶

¹College of Physical Science and Technology, Lingnan Normal
University, Zhanjiang, 524048, China.

²School of Mechatronic Engineering, Lingnan Normal University,
Zhanjiang, 524048, China.

³College of Chemistry and Chemical Engineering, Lingnan Normal
University, Zhanjiang, 524048, China.

⁴College of Computer Science and Intelligent Education, Lingnan
Normal University, Zhanjiang, 524048, China.

⁵School of Undergraduate Education, Shenzhen Polytechnic University,
Shenzhen, 518055, China.

⁶College of Physics and Electronic Information & Research Center for
Novel Solar-Blind Ultraviolet and Infrared Photoelectronic Detectors of
Henan Province, Luoyang Normal University, Luoyang, 471934, China.

*Corresponding author(s). E-mail(s): wuxm@lingnan.edu.cn;
z1797808078@163.com;

Keywords: Serverless computing, Cold start, Trace-driven simulation, Warm-budget
allocation, Supplementary information

Supplementary Information for Journal of Grid Computing. Online Resource 1
(ESM.1.pdf). Corresponding authors: Xuming Wu (wuxm@lingnan.edu.cn) and
Gaoyuan Zhao (z1797808078@163.com).

Table S1 Azure four-window robustness at $1 \times B_{ref}$ using request-weighted cold-start rate (CSR). Each 3-day window independently re-selects the top-1000 functions by total invocations and recomputes $B_{ref} = \text{Cost}(\text{GlobalTTL} = 5 \text{ min})$ within that window. Mean $\pm$ std uses the sample standard deviation over the four non-overlapping windows ($n = 4$).

Window	Global	IAT	Hist.	Full	BF
W1 (Days 1–3)	0.00595	0.00445	0.00485	0.00368	0.00374
W2 (Days 4–6)	0.00637	0.00478	0.00520	0.00401	0.00408
W3 (Days 7–9)	0.00614	0.00461	0.00501	0.00386	0.00392
W4 (Days 10–12)	0.00678	0.00500	0.00546	0.00405	0.00410
Mean $\pm$ std	0.00631 $\pm$ 0.00036	0.00471 $\pm$ 0.00024	0.00513 $\pm$ 0.00026	0.00390 $\pm$ 0.00007	0.00396 $\pm$ 0.00017

Table S2 Alibaba Microservices v2021 providerRPC-only robustness at $1 \times B_{ref}$ across five non-overlapping 2-h subwindows. Each subwindow independently re-selects the top-1000 services by total reconstructed providerRPC request count and recomputes its own B_{ref} . Mean $\pm$ std uses the sample standard deviation over the five windows ($n = 5$).

Window	Global	IAT-Mean	Histogram	Full greedy	BudgetFunnel
W1 (0–2 h)	0.01395	0.01071	0.01156	0.00830	0.00846
W2 (2–4 h)	0.01482	0.01138	0.01224	0.00879	0.00896
W3 (4–6 h)	0.01527	0.01172	0.01266	0.00909	0.00927
W4 (6–8 h)	0.01441	0.01106	0.01193	0.00860	0.00877
W5 (8–10 h)	0.01578	0.01209	0.01302	0.00943	0.00961
Mean $\pm$ std	0.01485 $\pm$ 0.00072	0.01139 $\pm$ 0.00054	0.01228 $\pm$ 0.00058	0.00884 $\pm$ 0.00041	0.00901 $\pm$ 0.00044

S1 Multi-window Robustness

Table S1 reports the Azure window results and Table S2 reports the Alibaba window results. We use provider-specific non-overlapping robustness windows. The Azure archive contains 14 consecutive 24-h invocation files, so the valid non-overlapping 3-day protocol comprises four windows: W1=Days 1–3, W2=Days 4–6, W3=Days 7–9, and W4=Days 10–12. Days 13–14 are reserved as a 2-day temporal hold-out and are excluded from the four-window aggregate. Alibaba Cluster Trace–Microservices v2021 has a 12-h MSRTQps horizon at 60-s granularity; the five robustness windows are W1=0–2 h, W2=2–4 h, W3=4–6 h, W4=6–8 h, and W5=8–10 h, with 10–12 h reserved as a hold-out/additional window. The same TTL grid and counting semantics are used within each provider.

Across the four valid 3-day Azure windows, BudgetFunnel remains close to full-evaluation greedy at $1 \times B_{ref}$ while preserving the same ordering relative to the three TTL baselines in every window. Days 13–14 remain reserved as a temporal hold-out and are not included in these summary statistics.

Across these five provider-specific robustness windows, BudgetFunnel remains close to full-evaluation greedy and preserves the ordering relative to the three TTL baselines.

The main Alibaba evaluation uses the full 12-h horizon; 10–12 h remains reserved as an additional temporal hold-out.

S2 Alibaba Workload Provenance and Preprocessing

The Azure Functions 2019 trace is the other public workload used by the main manuscript [1]. The main text provides the complete trace-selection protocol; this section records the Alibaba-specific preprocessing needed to reproduce Tables S1–S2. The cross-workload robustness dataset is *Alibaba Cluster Trace–Microservices v2021* from the official `alibaba/clusterdata` repository [2], version `cluster-trace-microservices-v2021`. We use the MSRTQps module and the audited file set `MSRTQps_0.csv–MSRTQps_24.csv`, which covers 12 h at 60-s granularity. Each `msname` is one allocation entity and records with the same `msname` and minute slot are summed over replicas (`msinstanceid`). A field-level audit found a shard-specific anomaly in `MSRTQps_0.csv`: `HTTP_MCR` equals `HTTP_RT` for 99.76% of valid records, compared with about 0.13% median exact equality across the remaining shards. We therefore use `providerRPC_MCR` only as the primary incoming-call signal. Since this field is measured in calls/s, the per-minute request-count proxy is reconstructed as

$$a_f(t) = \text{round} \left(60 \sum_{i \in I_f(t)} \text{providerRPC_MCR}_{f,i,t} \right).$$

Consumer-side RPC is excluded to avoid counting calls emitted by the service as incoming invocations, and `HTTP_MCR` is excluded from the primary signal because of the audited source anomaly. Invalid/NaN timestamps, non-positive reconstructed counts, and never-active services are removed. The top 1000 services are selected by $\sum_t a_f(t)$ over the full 12-h horizon, with ascending `msname` as the tie-break. Multiplication by the common factor 60 changes the absolute request-count total but, by itself, does not change request-weighted CSR, service ranking, or TTL allocation when the same signal composition is used throughout. This dataset is a microservice workload trace rather than a native FaaS trace; all Alibaba CSR/CSO values are therefore simulated robustness results under the common TTL abstraction, not measured Alibaba FaaS cold-start rates. Because the 12-h main horizon and 2-h robustness windows are shorter than the 24-h periodicity lag used by the proxy, the periodicity term is set to zero by the insufficient-horizon rule; Alibaba candidate screening therefore uses burstiness and non-sparsity only. Under the providerRPC-only reconstruction, the selected top-1000 workload contains approximately 4.96×10^8 reconstructed requests, with 30.9% average sparsity and a 69.1% active-slot ratio.

S3 Candidate-Ratio Ablation

Tables S3 and S4 report the Azure and Alibaba candidate-ratio ablations, respectively.

Table S3 Azure candidate-ratio ablation at $1 \times B_{ref}$. Recall is recall at the top 10% impact set; Gap is relative CSR gap versus $\gamma = 1$.

γ	Candidates	Recall	Precision	CSR	Gap	Total (s)	Speedup
0.05	50	0.47	0.94	0.00428	9.74%	0.20	14.8×
0.10	100	0.86	0.86	0.00396	1.54%	0.33	9.0×
0.20	200	0.96	0.48	0.00392	0.51%	0.63	4.7×
1.00	1000	1.00	0.10	0.00390	0	2.97	1.0×

Table S4 Alibaba candidate-ratio ablation at $1 \times B_{ref}$. Recall is recall at the top 10% impact set; Gap is relative CSR gap versus $\gamma = 1$.

γ	Candidates	Recall	Precision	CSR	Gap	Total (s)	Speedup
0.05	50	0.41	0.82	0.00988	11.89%	0.205	14.6×
0.10	100	0.81	0.81	0.00900	1.93%	0.333	9.0×
0.20	200	0.93	0.47	0.00889	0.68%	0.637	4.7×
1.00	1000	1.00	0.10	0.00883	0	2.990	1.0×

Table S5 Proxy feature ablation on Azure at $1 \times B_{ref}$ and $\gamma = 0.10$.

Proxy feature(s)	Recall@Top10%	CSR	Gap vs $\gamma = 1$
Burstiness only	0.71	0.00422	8.21%
Non-sparsity only	0.76	0.00414	6.15%
Periodicity only	0.39	0.00461	18.21%
Burstiness + non-sparsity	0.83	0.00401	2.82%
Burstiness + periodicity	0.75	0.00415	6.41%
Non-sparsity + periodicity	0.78	0.00410	5.13%
All three	0.86	0.00396	1.54%
Full evaluation	1.00	0.00390	0

S4 Proxy Feature Ablation

Table S5 reports the proxy-feature ablation.

S5 Cold-Counting Semantics

Table S6 compares request-weighted and slot-level cold-start rate.

The absolute rate depends on the aggregation semantics, but the ranking is identical.

Table S6 Request-weighted and slot-level CSR on Azure at $1 \times B_{ref}$.

Method	Request CSR	Slot CSR	Request rank	Slot rank
Global TTL	0.00630	0.0138	5	5
Histogram TTL	0.00511	0.0119	4	4
IAT-Mean TTL	0.00472	0.0108	3	3
BudgetFunnel	0.00396	0.0093	2	2
Full-evaluation greedy	0.00390	0.0091	1	1

Table S7 TTL-grid sensitivity on Azure at $1 \times B_{ref}$.

TTL grid (min)	CSR	Fine runtime (s)
{0, 0.5, 1, 2, 5, 10, 20, 30, 60}	0.00396	0.290
{0, 1, 2, 5, 10, 20, 30, 60}	0.00396	0.259
{0, 1, 5, 10, 30, 60}	0.00410	0.198
{0, 1, 2, 3, 5, 10, 15, 20, 30, 45, 60}	0.00391	0.354

Table S8 Fixed cold-penalty sensitivity, reported as cold-start overhead per request (CSO/Req, s/request), at $1 \times B_{ref}$.

Trace	L_c	Global	IAT-Mean	Full-eval	BudgetFunnel
Azure	1 s	0.00630	0.00472	0.00390	0.00396
Azure	5 s	0.03150	0.02360	0.01950	0.01980
Azure	10 s	0.06300	0.04720	0.03900	0.03960
Alibaba	1 s	0.01485	0.01140	0.00883	0.00900
Alibaba	5 s	0.07425	0.05700	0.04415	0.04500
Alibaba	10 s	0.14850	0.11400	0.08830	0.09000

S6 TTL-Grid Sensitivity

Table S7 reports the TTL-grid sensitivity.

S7 Cold-Penalty Sensitivity

Tables S8 and S9 report fixed and heterogeneous cold-penalty sensitivity, respectively.

For each σ , we set $\mu = \ln(10) - \sigma^2/2$, so the untruncated log-normal distribution has mean 10 s. Penalties are then clipped to [1,60] s and seed 42 is reused across methods. Because clipping changes the realized empirical mean, especially at larger σ , this is interpreted as a combined tail-shape/clipping sensitivity analysis rather than a fixed-post-clipping-mean comparison.

Table S9 Heterogeneous log-normal cold-penalty sensitivity on Azure, reported as CSO/Req (s/request), at $1 \times B_{ref}$.

Trace	σ	Global	IAT-Mean	Full-eval	BudgetFunnel
Azure	0.5	0.0628	0.0470	0.0389	0.0395
Azure	1.0	0.0606	0.0454	0.0373	0.0378
Azure	1.5	0.0523	0.0393	0.0324	0.0329

Table S10 Same-scope exact mixed-integer linear program (MILP) optimum versus greedy on reduced Azure workloads.

N	Exact	CSR	Greedy	CSR	Gap	Status	MILP runtime	Greedy runtime
50	0.00281		0.00285	1.42%	OPTIMAL		0.09 s	0.001 s
100	0.00312		0.00318	1.92%	OPTIMAL		0.31 s	0.003 s
200	0.00346		0.00355	2.60%	OPTIMAL		1.87 s	0.007 s

S8 Same-Scope Exact MILP Validation

Table S10 reports the exact MILP comparison. For each $N \in \{50, 100, 200\}$, we select the top- N functions by total invocations from the same preprocessed Azure top-1000 workload, breaking ties by ascending function ID. The reference budget is recomputed inside each reduced scope as

$$B_{ref,N} = Cost_{\mathcal{F}_N}(\text{Global TTL} = 5 \text{ min}),$$

and both methods are compared at $B = 1 \times B_{ref,N}$. CP-SAT settings are OR-Tools 9.10.4067, 60-s time limit, `num_search_workers=1`, relative gap limit 0, and random seed 42. Every audited MILP run returns **OPTIMAL**.

The exact solver and greedy method use identical functions, TTL grids, simulated cost/benefit tables, and reduced-scope budgets. The reported gap is therefore the empirical greedy optimality gap for the evaluated 0–1 formulations.

S9 Reference-Budget Sensitivity

Table S11 reports the reference-budget sensitivity.

For the audited Azure representative window, the default 5-min reference therefore corresponds to 2.47×10^7 warm instance-seconds. The absolute reference costs change with the reference TTL, while the BudgetFunnel–full-evaluation CSR gap remains stable (1.42–1.54%) across the three normalizations.

Table S11 Sensitivity to the reference TTL used to define B_{ref} on Azure. Here B_{ref} is in warm instance-seconds; BF denotes BudgetFunnel.

Ref. TTL	B_{ref} (s)	Global	IAT	Hist.	Full	BF	Gap
1 min	1.18×10^7	0.01095	0.00820	0.00874	0.00675	0.00685	1.48%
5 min	2.47×10^7	0.00630	0.00472	0.00511	0.00390	0.00396	1.54%
10 min	3.86×10^7	0.00455	0.00346	0.00368	0.00282	0.00286	1.42%

Table S12 Alibaba complete budget sweep using request-weighted CSR.

B/B_{ref}	Global	IAT-Mean	Histogram	Full-eval	BudgetFunnel
0.5	0.02290	0.01800	0.01925	0.01435	0.01466
1	0.01485	0.01140	0.01232	0.00883	0.00900
2	0.00948	0.00739	0.00787	0.00568	0.00578
5	0.00432	0.00356	0.00375	0.00305	0.00310
10	0.00272	0.00228	0.00236	0.00192	0.00195
20	0.00203	0.00177	0.00181	0.00157	0.00159
30	0.00186	0.00163	0.00167	0.00148	0.00150
50	0.00182	0.00160	0.00164	0.00145	0.00147

Table S13 Azure BudgetFunnel and full-evaluation greedy complete budget sweep.

B/B_{ref}	Full-eval CSR	BudgetFunnel CSR
0.5	0.00605	0.00620
1	0.00390	0.00396
2	0.00235	0.00239
5	0.00118	0.00120
10	0.00084	0.00085
20	0.00072	0.00073
30	0.00069	0.00070
50	0.00068	0.00069

S10 Complete Budget Sweeps

Tables S12 and S13 report the Alibaba and Azure complete budget sweeps, respectively.

The maximum relative BudgetFunnel gap to full evaluation is 2.48% on Azure and 2.16% on Alibaba over these sweeps.

S11 Matched Resource-Utilization Diagnostics

Table S14 reports the matched resource-utilization diagnostics.

Table S14 Azure WMT/Cost and ActualCost/ B at matched budgets, in percent. WMT denotes wasted warm time; columns 1, 5, and 10 denote B/B_{ref} .

Method	WMT 1×	WMT 5×	WMT 10×	Cost/ B 1×	Cost/ B 5×	Cost/ B 10×
Global TTL	69.8	81.5	87.9	100.0	96.8	82.6
IAT-Mean TTL	66.4	78.3	84.9	99.0	95.6	80.8
Histogram TTL	68.7	80.6	86.5	98.7	95.1	81.4
Full-evaluation greedy	61.9	73.8	80.1	99.5	95.7	79.8
BudgetFunnel	62.8	74.6	80.9	99.1	94.8	78.6

Table S15 Runtime breakdown in seconds, reported as mean $\pm$ std.

Scale	γ	Proxy	Fine	Allocation	Total	Speedup
top-1k	0.10	0.031 $\pm$ 0.002	0.294 $\pm$ 0.009	0.004 $\pm$ 0.001	0.329 $\pm$ 0.010	9.0×
top-1k	1.00	0.031 $\pm$ 0.002	2.921 $\pm$ 0.061	0.015 $\pm$ 0.002	2.967 $\pm$ 0.063	1×
top-5k	0.10	0.143 $\pm$ 0.005	1.455 $\pm$ 0.027	0.013 $\pm$ 0.002	1.611 $\pm$ 0.030	9.0×
top-5k	1.00	0.143 $\pm$ 0.005	14.349 $\pm$ 0.181	0.075 $\pm$ 0.004	14.567 $\pm$ 0.185	1×

BudgetFunnel differs from full-evaluation greedy by less than one percentage point in WMT/Cost at all three budgets (0.9, 0.8, and 0.8 points), while maintaining similar budget utilization. Its WMT/Cost is also lower than Global TTL, IAT-Mean TTL, and Histogram TTL in all three settings.

S12 Runtime and Reproducibility

Table S15 reports the runtime breakdown.

Experiments were conducted on an Intel Xeon E5-2620 v4 @ 2.10 GHz machine with 8 physical cores (16 logical threads), 32 GB RAM, and Ubuntu 20.04.6 LTS x86_64. The environment used Python 3.9.18, NumPy 1.26.4, Pandas 2.1.4, and Google OR-Tools 9.10.4067. Runtime entries report mean $\pm$ std over 10 runs. Main experiments used one execution thread; CP-SAT used `num_search_workers=1`. The primary trace granularity is 60 s, the default TTL grid is $\{0, 0.5, 1, 2, 5, 10, 20, 30, 60\}$ min, and the full budget grid is $\{0.5, 1, 2, 5, 10, 20, 30, 50\} \times B_{ref}$. Public provider traces should be obtained from their original repositories. **Supplementary Code S1** provides the essential reference implementation of BudgetFunnel, including the trace-driven TTL simulator, proxy screening, precedence-aware budget allocation, same-scope MILP validation, preprocessing/data-audit utilities, and documented runtime/MILP configurations. Raw provider traces are not redistributed and should be obtained from the public repositories cited in the manuscript.

References

- [1] Microsoft: Azure Functions Trace 2019 (AzurePublicDataset). <https://github>.

com/Azure/AzurePublicDataset/blob/master/AzureFunctionsDataset2019.md.
Accessed 16 January 2026 (2019)

- [2] Alibaba: Alibaba Cluster Trace-Microservices v2021 (clusterdata). <https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2021>.
Accessed 16 January 2026 (2021)