

HKD Infinity for Prompt-Conditioned Video Production: Exactness, Conditional Cycle Superiority, and Measured Benchmarks

Michael S Yang
Independent Researcher
yangofzeal@gmail.com

October 17, 2026

Abstract

We establish an exactness theorem for deterministic prompt-conditioned video computation with semantics-preserving state reuse, a lossless prompt–residual reconstruction theorem, and an explicit sufficient condition for at least $1000\times$ hardware-cycle superiority over a specified comparator. These are conditional mathematical statements, not a proof that the current HKD implementation beats Runway or all state-of-the-art systems. Available empirical evidence consists of a single matched self-hosted CogVideo kettle pair: 659.412 seconds with reference validation versus 644.553 seconds with HKD validation and dispatch, a wall-time ratio of 1.02305. The decoded RGB hashes agree. No matched Runway generation, provider cycle counters, successful three-to-five-minute training run, or universal face/body fidelity result is available. We give the actual benchmark driver and the executable subset-sum verification code, distinguish counted operations from hardware cycles, and state precisely what additional evidence a stronger claim requires.

1 Scope and relation to HKD

The subset-sum HKD paper studies width-controlled state exploration and reports selected operation-count improvements on structured examples [1]. Its retained-width bound motivates reducing redundant work, but does not by itself establish correctness of every pruning rule or superiority of neural video generation. The HKD transfer–distillation paper makes the relevant exactness requirement explicit: merged residual states must preserve their completion semantics; its universal polynomial-state bound is unresolved [2]. Here we transfer this *proof obligation*, rather than a numerical speedup claim, to video computation.

The concrete renderer examined here is self-hosted `hkd_render3.py` with CogVideoX-2b. The located HKD components provide bounded dispatch, CPU visual validation, and measurement or coordination functions. A separately identifiable HKD-INF neural video accelerator has not been established. Diffusers documents memory/offload strategies that affect the real inference path [3]. An HKD name or wrapper does not replace the work of that model.

2 Formal video-production model

Fix frame count F , height H , width W , channel count C , and b bits per channel. A decoded video is $V \in \{0, 1\}^B$, where $B = FHWCb$. Let

$$Y = G_{\vartheta, \eta}(p, z)$$

be a deterministic finite computation producing B bits from prompt p and seed z . The specification η includes tokenizer, model weights, arithmetic, operation order, random-generator state, software, preprocessing, and output conversion. A fixed seed alone does not imply cross-platform bitwise reproducibility.

We distinguish (i) equality of decoded pixels with a reference computation; (ii) lossless reconstruction of a specified target; (iii) semantic prompt fidelity, including correct anatomy and motion; and (iv) byte equality of an encoded MP4. Neither (i) nor (ii) alone proves (iii). Identical decoded pixels need not yield identical MP4 bytes because encoding or metadata can differ.

3 Exact video computation and a cycle-superiority theorem

Theorem 1 (Semantics-preserving HKD quotient evaluation). *Let the complete computation of $G_{\vartheta, \eta}(p, z)$ be represented by a finite directed acyclic graph. Permit two nodes to share one evaluation only if their operations, complete inputs, and relevant execution states are identical and deterministic. Evaluate every canonical node once, preserving dependency order and the reference operation semantics. Then HKD quotient evaluation produces exactly the same decoded video as the unmerged computation.*

Proof. Induct over a topological order. Source nodes have the same specified values. At any subsequent node, the induction hypothesis gives identical dependency values. Applying the same deterministic operation with the same execution state yields the same result. Replacing a repeated computation by a reference to this result therefore changes no successor value. The output pixels consequently agree bit for bit. $\square$

Floating-point reassociation, quantization, approximate feature matching, or deletion of non-identical diffusion states is not covered. Independent transitions may run concurrently only when the schedule does not change their semantics. Kernel delta updates require an exact algebraic or machine-arithmetic justification; the label “delta” is insufficient.

Theorem 2 (Instancewise certified cycle superiority). *Fix a hardware and execution accounting model and a specified task with identical required outputs. Suppose the HKD execution has Q canonical evaluations and E dependency/reference operations. Let c, d be valid upper bounds on the respective costs, and let A upper-bound all remaining cycle costs, including canonicalization, memory, scheduling, synchronization, input/output, residual processing, and encoding. For a named comparator S , suppose $C_S \geq L > 0$. Then*

$$C_H \leq U := cQ + dE + A, \quad \frac{C_S}{C_H} \geq \frac{L}{U}$$

In particular, $L \geq 1000U$ implies at least $1000\times$ cycle superiority while retaining the preceding exactness guarantee.

Proof. Summing the bounded HKD costs gives $C_H \leq U$. Dividing the comparator lower bound by this positive upper bound yields the result. $\square$

The theorem is useful only when its bounds are independently established. No measured $L \geq 1000U$ has been established for the current renderer versus Runway. To extend the conclusion to a class of “SOTA” algorithms, the lower bound must hold for every member of that precisely defined class. A comparator can itself use memoization. CPU cycles, summed GPU cycles, and elapsed GPU time are different quantities and must not be interchanged.

Corollary 3 (A concrete conditional $1000\times$ family). *Suppose a specified recomputing baseline performs at least 2000 identical kernel evaluations, each costing at least $M > 0$ cycles. Suppose exact HKD reuse evaluates the kernel once and completes all remaining work within a total budget of $2M$ cycles. Then its cycle speedup over that baseline is at least 1000.*

Proof. The baseline costs at least $2000M$, whereas HKD costs at most $2M$. $\square$

This exhibits a sufficient structural regime; it is not an empirical CogVideo result or a lower bound against the best caching algorithm. If both systems produce the exact reference, exactness is *parity*. Strict accuracy superiority additionally requires a nonzero comparator error under the same reference and metric.

4 Exact prompt-based encoding

Theorem 4 (Prompt-residual lossless reconstruction). *For any target V , define $Y = G_{\vartheta,\eta}(p, z)$ and $R = V \oplus Y$. Let L_c be a lossless residual codec. Encode $(\vartheta, \eta, p, z, L_c(R))$, with any pre-shared components explicitly declared. Then decoding by*

$$D = G_{\vartheta,\eta}(p, z) \oplus L_c^{-1}(L_c(R))$$

returns V exactly. If $R = 0$, a zero-residual flag suffices.

Proof. Lossless decoding returns R ; determinism reproduces Y ; hence $D = Y \oplus (V \oplus Y) = V$. $\square$

This standard predictive-residual construction is a rigorous way to interpret exact prompt-based encoding. It is not a claim of a novel XOR identity, a short residual, or perfect text-to-video synthesis. An MP4 delivery must use a suitable lossless path if exact decoded target pixels are promised; a lossy encoding stage generally breaks that promise.

Theorem 5 (No universal short-prompt lossless encoding). *For Fixed video length B , any injective encoding of every B -bit video has a worst-case description length of at least B bits, counting all non-pre-shared prompts, seeds, residuals, and model changes.*

Proof. There are 2^B videos. Binary descriptions of length at most $\ell < B$ number $2^{\ell+1} - 1 < 2^B$. Two videos would receive the same description; a deterministic decoder cannot reconstruct both. Therefore no such universal shorter encoding exists. $\square$

The theorem permits substantial compression of structured videos. It prohibits calling an omitted residual or a stored-output pointer an exact encoding of arbitrary unseen content without accounting for the necessary information.

5 Exact subset-sum connection and a failure certificate

The exact recurrence

$$\text{SAT}(i, t) = \text{SAT}(i + 1, t) \vee \text{SAT}(i + 1, t - a_i)$$

with correct terminal conditions permits canonical reuse of (i, t) . Correctness follows by splitting solutions according to whether they include a_i , then inducting over remaining items. The number of states can remain large; a numeric-target bound is not automatically polynomial in binary input length.

Nearest-target width pruning is different. On $[8, 7, 6]$ with target 13 and width 1, keeping only the largest positive partial sum retains 8, then discards 7. It misses $7 + 6 = 13$, even if zero is also retained. This refutes universal exactness of that pruning rule at this width, not all HKD algorithms. Appendix C contains the executed example and 256 seeded comparisons of exact memoization with exhaustive enumeration. All 256 comparisons and five doctest examples passed; these are mathematical implementation checks, not video benchmarks.

6 Available empirical video results

The historical kettle benchmark used two fresh CogVideoX-2b generations with the same prompt, seed 42, 30 diffusion steps, and 49 native frames, delivered as five-second, 1280×720 , 24-fps MP4s. The experimental switch enabled HKD CPU validation/dispatch; it did not substitute a different inference model. The actual driver appears in Appendix B. Results were read from its existing remote status.json; they were not regenerated for this paper. Local records and source provenance are listed in Appendix A.

Table 1: Historical single-pair kettle measurements. Times are wall seconds, not cycles.

Measurement	Reference mode	HKD mode
Render subprocess elapsed	659.412120	644.552855
Post-render frame classification	0.459723	0.114076
Lossless codec round trip	0.539615	0.704379
Visual-guard checked frames	120	120
Blank/grid failures	0	0
MP4 file bytes	389590	389639

Both modes recorded classifier parity against the reference and the same decoded RGB SHA-256:

5aef7d48a8ca4d1ab44b36a35208fcc3db214ec082d8d2f7bb0014d9c7af7973

The differing MP4 sizes illustrate why decoded equality and encoded-byte equality are separate. Hash agreement is strong empirical integrity evidence, not a universal proof of collision-free equality. The recorded visual gate checks content/blank/grid failures, not perfect kettle motion or human anatomy.

6.1 Discussion of the benchmark speedup

The render-time ratio is

$$S_{\text{wall}} = 659.4121198609937 / 644.5528549458832 = 1.0230536.$$

The observed difference is 14.8593 seconds: approximately 2.25% lower elapsed time, or 2.31% greater reciprocal-time throughput for this single pair. These percentages have different denominators. One ordered pair cannot separate the treatment from warm caches, runtime noise, or other temporal effects; it supports neither statistical superiority nor causal attribution of the entire difference to HKD.

The separately timed classifier was approximately $4.03\times$ faster, saving 0.346 seconds. The codec round trip was slower in HKD mode by approximately 0.165 seconds. Those post-render component timings must not be added to, or interpreted as explaining, the entire render-time difference. The timer in the driver surrounds the rendering subprocess; later full raw decode, supplementary classification, and codec checks occur outside that interval. Neither column is a measured customer-download completion time.

For a fraction f of total runtime accelerated by factor r , unchanged remaining work gives

$$S = \frac{1}{(1-f) + f/r} \leq \frac{1}{1-f}.$$

Even infinite acceleration requires $f \geq 0.999$ to reach $1000\times$. Optimizing a small validation component cannot establish that result. The observed self-hosted $1.023\times$ ratio is about 978 times smaller than the requested $1000\times$ ratio; this is a comparison of ratios, not a forecast of engineering effort.

A separate recovered CGI turntable run recorded 646.997867 seconds, a five-second MP4, 120 checked frames, and full decode success. It is a different scene and not a matched comparator. No face/body quality result follows from it. The October 17 model-offload trial was still in progress at the analysis cutoff and is excluded from the completed-results table.

6.2 Runway and hardware-cycle limitations

Runway documents a no-credit Unlimited Mode for eligible accounts and supported models [4]. Account eligibility could not be inspected because browser access was unavailable. No Runway job or credit expenditure was made in this work. There is therefore no matched Runway MP4, latency measurement, or pixel comparison in this paper. Provider hardware-cycle counters are unavailable, and wall-clock latency cannot establish cycle superiority. A single provider is not the entire state of the art.

7 Training-time analysis

CogVideo training guidance describes pretrained-model adaptation and multi-video recipes [5]; it does not establish that one clip yields perfect new faces and motions within five minutes. No training or optimizer-step benchmark was executed for this paper. The observed Biggie hardware is a Tesla T4 with 15,360 MiB of GPU memory.

For complete optimizer updates S with measured time t_{step} , use

$$T_{\text{total}} = T_{\text{decode}} + T_{\text{load}} + T_{\text{cache}} + S t_{\text{step}} + T_{\text{save}} + T_{\text{validation}}.$$

A training-only timer may exclude validation only when explicitly labeled. As a planning example, assume 60 seconds of fixed overhead and initially exclude validation. At $t_{\text{step}} = 2, 10, 30$ seconds, a three-minute limit permits at most 60, 12, 4 updates; a five-minute limit permits 120, 24, 8. These are hypothetical arithmetic budgets, not measured training rates or convergence guarantees. The historical 644.6-second validation render alone exceeds five minutes in that configuration.

Proposition 6 (One-clip non-identifiability). *Without restrictions on the target function, one observed training clip cannot guarantee exact output on every unseen request, regardless of training duration.*

Proof. Choose two target functions agreeing on the observed example but differing on an unseen request. A learner receives identical evidence in both worlds and has the same output distribution. It cannot be correct with probability one for both different unseen targets. $\square$

Pretraining supplies useful priors but does not remove this logical limitation. Grayscale data also cannot uniquely determine arbitrary original colors: a mapping from 24-bit RGB to 8-bit grayscale is many-to-one. A three-to-five-minute adaptation experiment can be proposed, but its runtime, training gain, held-out motion quality, and memory use must be measured. No such successful training result is claimed here.

8 Conclusion

The strongest established video statement is conditional: exact canonical reuse preserves a specified deterministic reference, and certified work/cycle bounds imply quantitative superiority when their inequality holds. Prompt–residual encoding reconstructs any specified target exactly but need not compress it. Available completed rendering evidence is one approximately $1.023\times$ wall-time pair with matching decoded hashes, not $1000\times$ superiority over Runway, not hardware-cycle superiority, and not proof of perfect faces or motion. Replacing these qualifications with unconditional claims would exceed both the proofs and the measurements.

References

- [1] M. S. Yang, “Hilbert–Krylov Tower Decomposition and a Pseudo-Polynomial Complexity Bound for Subset Sum,” *International Journal of Computer Techniques*, vol. 12, no. 6, pp. 498–503, 2025. <https://ijctjournal.org/hilbert-krylov-pseudo-polynomial-complexity/>.
- [2] M. S. Yang, “HKD Infinity Transfer–Distillation Reduction for Exact Cover,” *International Journal of Computer Techniques*, vol. 13, no. 5, pp. 345–351, 2026. <https://ijctjournal.org/hkd-transfer-distillation-reduction-exact-cover/>.
- [3] Hugging Face, “Reduce memory usage,” Diffusers documentation, accessed October 17, 2026. <https://huggingface.co/docs/diffusers/optimization/memory>.
- [4] Runway, “Generating in Unlimited Mode,” documentation, accessed October 17, 2026. <https://help.runwayml.com/hc/en-us/articles/54428296933011-Generating-in-Unlimited-Mode>.
- [5] Hugging Face, “CogVideoX,” Diffusers training documentation, accessed October 17, 2026. <https://huggingface.co/docs/diffusers/training/cogvideox>.

A Benchmark provenance and reproduction limits

The kettle driver was retrieved from

`/home/ubuntu/hkd-render-biggie/jobs/kettle_benchmark_20261012/run.py`

with measurements from the sibling status.json. The retrieved driver is saved locally as

session_checks/render_recalibration_20261017/prior_runner.py.txt.

Its dependencies include the actual Render3/CogVideo backend, cached model weights, davids_sling, hkd_sling_codec, hkd_kernel, visual_guard, and FFmpeg. The listing is the executed historical driver, not a self-contained neural model implementation. The environment and hardware must be recorded for a fresh reproduction, and existing output directories must not be overwritten. The script's original two-second lock-acquisition retry is reproduced for provenance, not recommended as a current health-polling cadence.

The completed turntable observation is stored in

session_checks/cogvideo_live_fix_20261012/summary.json.

The subset verification output is stored in

session_checks/hkd_theorem_analysis_20261017/verification.json.

No training source, optimizer timing, or Runway benchmark code is presented as executed because none was executed for those tasks. Appendix code listings preserve the available benchmark and verification sources rather than manufacturing a 1000× result.

B Actual historical kettle benchmark driver

```

1  """Two fresh, matched CogVideo generations; no cached-output speedup claims."""
2  from pathlib import Path
3  import fcntl
4  import hashlib
5  import json
6  import os
7  import signal
8  import subprocess
9  import sys
10 import time
11
12  ROOT = Path('/home/ubuntu/hkd-render-biggie')
13  WORK = ROOT / 'jobs/kettle_benchmark_20261012'
14  PYTHON = ROOT / '.venv-py311/bin/python'
15  sys.path.insert(0, str(ROOT))
16  from davids_sling import HKD_ALU, HKD_KERNEL, HKD_THREAD
17  from hkd_sling_codec import exact_cover_stream, decode
18  from hkd_kernel import configured_kernel
19  from visual_guard import validate_video, classify_rgb
20
21  PROMPT = ('A continuous cinematic product shot of one bright polished stainless steel tea kettle '
22           'with a curved spout, round lid and black handle on a clean pale kitchen counter. '
23           'The kettle slowly rotates clockwise while the camera smoothly orbits from a front '
24           'three-quarter view to a side view, revealing changing perspective and realistic parallax. '
25           'A fine plume of steam curls out of the spout. Bright soft studio daylight, clear object '
26           'edges, physically coherent reflections, stable kettle geometry, no cuts, no people, '
27           'no text, no colored grid or mosaic.')
28
29
30  def save(report):
31      temporary = WORK / 'status.tmp'
32      temporary.write_text(json.dumps(report, indent=2))
33      temporary.replace(WORK / 'status.json')
34
35
36  def run():
37      report = dict(started=time.time(), state='waiting_for_gpu', trials=[],

```

```

38         prompt=PROMPT, seconds=5, size='1280x720', fps=24, seed=42, steps=30,
39         native_frames=49, model='THUDM/CogVideoX-2b',
40         comparator_scope='Same CogVideo inference, HKD validation/dispatch off versus on',
41         sota_status='not established', hardware_cycles='not measured',
42         inference_acceleration_claim=False, fresh_generations=True,
43         trial_count_per_mode=1, conclusion_scope='preliminary single pair, no statistical
superiority claim') save(report)
44     with (WORK / 'benchmark.lock').open('a') as single:
45         fcntl.flock(single, fcntl.LOCK_EX | fcntl.LOCK_NB)
46         with (ROOT / 'cogvideo.lock').open('a') as gpu, Path('/biggie2/biggie2/.spindrop-gpu.lock').open('a')
47     as audio:
48         deadline = time.monotonic() + 120
49         while True:
50             try:
51                 fcntl.flock(gpu, fcntl.LOCK_EX | fcntl.LOCK_NB)
52                 fcntl.flock(audio, fcntl.LOCK_EX | fcntl.LOCK_NB)
53                 break
54             except BlockingIOError:
55                 fcntl.flock(gpu, fcntl.LOCK_UN)
56                 if time.monotonic() >= deadline:
57                     raise RuntimeError('GPU remains busy; no concurrent model launch')
58                 time.sleep(2)
59     for mode in ('baseline', 'hkd'):
60         trial = dict(mode=mode, started=time.time(), state='running')
61         report['trials'].append(trial)
62         report['state'] = 'rendering_' + mode
63         save(report)
64         folder = WORK / mode
65         folder.mkdir(exist_ok=True)
66         backend = (str(PYTHON) + ' ' + str(ROOT / 'jobs/well_lit_cgi_1000_20261012/backend.py')
67                   + ' --prompt {prompt} --output {output} --seconds {seconds}'
68                   + ' --size {size} --fps {fps} --steps 30 --seed 42'
69                   + ' --progress-file ' + str(folder / 'progress.json'))
70         env = dict(os.environ, LD_LIBRARY_PATH=str(ROOT / 'sqlite-3.51.3'),
71                   HKD_KERNEL='off' if mode == 'baseline' else 'on',
72                   HF_HUB_OFFLINE='1', TRANSFORMERS_OFFLINE='1')
73         output = folder / 'kettle.mp4'
74         command = [str(ROOT / 'hkd_render3.py'), 'render', PROMPT,
75                   str(output), '--seconds', '5', '--size', '1280x720',
76                   '--fps', '24', '--backend', backend]
77         start = time.perf_counter()
78         with (folder / 'render.log').open('w') as log:
79             process = subprocess.Popen(command, cwd=str(ROOT), env=env,
80                                       stdout=log, stderr=subprocess.STDOUT, start_new_session=True)
81             try:
82                 rc = process.wait(timeout=1800)
83             except subprocess.TimeoutExpired:
84                 os.killpg(process.pid, signal.SIGTERM)
85             try:
86                 process.wait(timeout=15)
87             except subprocess.TimeoutExpired:
88                 os.killpg(process.pid, signal.SIGKILL)
89                 process.wait()
90             rc = 124
91         trial['render_wall_seconds'] = time.perf_counter() - start
92         trial['exit_code'] = rc
93         if rc:
94             trial.update(state='failed', finished=time.time())
95             save(report)
96             continue
97         trial['file_sha256'] = hashlib.sha256(output.read_bytes()).hexdigest()
98         trial['file_bytes'] = output.stat().st_size
99         trial['visual_guard'] = validate_video(output, kernel=configured_kernel())
100        raw = subprocess.check_output(['ffmpeg', '-v', 'error', '-xerror', '-i', str(output),
101                                      '-pix_fmt', 'rgb24', '-f', 'rawvideo', '-'], timeout=60)
102        trial['decoded_rgb_sha256'] = hashlib.sha256(raw).hexdigest()
103        del raw

```

```

104         small = subprocess.check_output(['ffmpeg', '-v', 'error', '-i', str(output), '-vf',
105                                         'scale=64:36', '-pix_fmt', 'rgb24', '-f', 'rawvideo', '-'])
106         frames = [small[i:i+6912] for i in range(0, len(small), 6912)]
107         assert len(frames) == 120 and all(len(frame) == 6912 for frame in frames)
108         kernel = configured_kernel()
109         begin = time.perf_counter()
110         labels = (HKD_THREAD(len(frames), list(range(len(frames))),
111                             lambda i: kernel.classify(frames[i]), workers=2)
112                  if mode == 'hkd' else [classify_rgb(frame) for frame in frames])
113         trial['classification_wall_seconds'] = time.perf_counter() - begin
114         reference = HKD_ALU(len(frames), range(len(frames)), lambda i: classify_rgb(frames[i]))
115         assert labels == reference
116         failures = 0
117         for label in labels:
118             failures = HKD_KERNEL(failures, 0, int(label != 'content'))
119         assert failures == 0
120         trial['classifier_parity'] = True
121         trial['failed_frames'] = failures
122         data = output.read_bytes()
123         phrases = [b'\x00'*8, b'\x00'*4, b'fyp', b'moov', b'mdat', b'avc1']
124         begin = time.perf_counter()
125         encoded, references = exact_cover_stream(data, phrases)
126         restored = decode(encoded, phrases)
127         trial['codec_roundtrip_seconds'] = time.perf_counter() - begin
128         assert restored == data
129         trial['codec'] = dict(exact_roundtrip=True, encoded_bytes=len(encoded),
130                             dictionary_payload_bytes=sum(map(len, phrases)),
131                             dictionary_framing_not_included=True, references=references,
132                             inference_speedup=False)
133         trial.update(state='validated_pending_visual_motion_review', finished=time.time()) save(report)
134         trials = report['trials']
135         if all(t['exit_code'] == 0 for t in trials):
136             report['render_wall_speedup'] = trials[0]['render_wall_seconds'] / trials[1]['
137             render_wall_seconds']
138             report['decoded_frame_byte_equality'] = trials[0]['decoded_rgb_sha256'] == trials[1]['
139             decoded_rgb_sha256']
140             report.update(state='finished_pending_review', finished=time.time())
141             save(report)
142         if __name__ == '__main__':
143             try:
144                 run()
145             except Exception as error:
146                 path = WORK / 'status.json'
147                 report = json.loads(path.read_text()) if path.exists() else {}
148                 report.update(state='failed', error=str(error), finished=time.time()) save(report)
149                 raise
150
151

```

C Executed exactness and budget verification

```

1  """Executable checks accompanying the original conditional theorem.
2
3  >>> exact([8, 7, 6], 13)[0]
4  True
5  >>> beam([8, 7, 6], 13, 1)
6  False
7  >>> exact([8, 7, 6], 12)[0]
8  False
9  >>> budget_steps(300, 60, 10)
10 24
11 >>> budget_steps(180, 200, 10)
12 0

```

```

13 """
14 from functools import lru_cache
15
16 def exact(values, target):
17     """Canonical (index, remaining target) DAG; no approximate pruning."""
18     values = tuple(values)
19     if target < 0 or any(x <= 0 for x in values):
20         raise ValueError('positive input values and nonnegative target')
21     @lru_cache(None)
22     def visit(i, remaining):
23         if remaining == 0:
24             return True
25         if i == len(values) or remaining < 0:
26             return False
27         return visit(i + 1, remaining) or visit(i + 1, remaining-values[i])
28     answer = visit(0, target)
29     return answer, visit.cache_info().currsz
30
31 def beam(values, target, width):
32     """Closest-to-target width cap, retaining zero as in a paper code branch."""
33     states = {0}
34     for value in sorted(values, reverse=True):
35         candidates = states | {s+value for s in states if s+value <= target}
36         if target in candidates:
37             return True
38         states = set(sorted(candidates, reverse=True)[:width]) | {0}
39     return False
40
41 def budget_steps(seconds, overhead, step_seconds):
42     return max(0, int((seconds-overhead)//step_seconds))
43
44 if __name__ == '__main__':
45     import doctest, itertools, json, random
46     result = doctest.testmod()
47     assert result.failed == 0
48     rng = random.Random(20261017)
49     for _ in range(256):
50         a = [rng.randrange(1, 30) for _ in range(rng.randrange(1, 11))]
51         target = rng.randrange(sum(a)+2)
52         reference = any(sum(x for x,bit in zip(a,bits) if bit)==target
53                        for bits in itertools.product([0,1],repeat=len(a)))
54         assert exact(a,target)[0] == reference
55     print(json.dumps({'doctest_examples':result.attempted,'failures':0,
56                     'exhaustive_reference_random_instances':256,
57                     'counterexample':{'values':[8,7,6],'target':13,'width':1,
58                                     'exact':True,'beam':False},
59                     'hardware_cycle_measurement':False}))

```

Recorded output:

```

1 {"doctest_examples": 5, "failures": 0, "exhaustive_reference_random_instances": 256, "counterexample": {"
  values": [8, 7, 6], "target": 13, "width": 1, "exact": true, "beam": false}, "hardware_cycle_measurement
  ": false}

```