nvpack

En arkivpackare som **vet vad den packar** — och som mäter sig mot allt annat som finns installerat på maskinen. Fyra riktiga filträd, tolv packare, fyrtiosju rundgångar, **varje enskild bit-exakt**.

PROGRAMKOD	BILDER (PNG)	FOTON (JPEG)	BLANDAT PROJEKTTRÄD
−37 %	**−42 %**	**−15 %**	**−38 %**
11,5 → 7,2 MiB 13 % under xz −9e	362,9 → 208,9 MiB 15 % under zstd −22	406,3 → 345,1 MiB 15 % under bzip2 −9	452,8 → 279,1 MiB 6 % under xz −9e

Mätt på en Ryzen AI 9 HX 470 (24 trådar, 2,0 GHz) med NVMe-disk.
Koden, korpusbyggaren och alla mätvärden ligger i `~/nvpack`.

SAMMANFATTNING

Vad problemet är

`tar | zstd` är två program som inte vet något om varandra. tar lägger filerna i den ordning katalogträdet råkar ha dem och skickar vidare en enda byteström. zstd ser den strömmen genom ett fönster och får aldrig veta att fil 4 000 är en exakt kopia av fil 12, att de 9 MB "slump" den just gav upp på i själva verket är en deflate-ström som går att veckla ut, eller att de 400 MB foton den maler igenom har ett entropilager från 1992 som går att byta ut.

nvpack fyller tomrummet mellan behållaren och komprimeraren. Den klassar innehållet, byter ut representationen där det finns en bättre, grupperar likt med likt, och väljer kodek per block genom att faktiskt prova. Fem mekanismer — och den här rapporten redovisar vad var och en gav, inklusive de som gav noll.

Mekanismerna, och vad de faktiskt bidrog med

MEKANISM	UTFALL	MÄTT PÅ
Omvandla innehållet	det enda som verkligen bär: **−15 %** på foton, **−42 %** på PNG	foto, png
Dedup på filnivå	stort eller noll, aldrig mittemellan: 28,7 % dubbletter i PNG-korpusen, 0,7 % i kodkorpusen	alla fyra
Sortera likt med likt	**−1,85 %** … men registret kostar **+1,81 %** → netto noll	kod
Välja kodek genom prov	hela stegen zstd −19 → xz → nvcm: 8,53 → 8,23 → 7,19 MiB	kod
Prova i stället för anta	**175 MiB** på PNG-korpusen, och gränsen mellan vinst och förlust på ett riktigt arkiv	png, skarpt arkiv

Sorteringsraden är värd att stanna vid. Den lät som en av huvudidéerna och visade sig betala exakt för sitt eget index, inte mer. Vinsterna kommer från att data *byter representation* före komprimeringen — inte från att den ligger i en smartare ordning.

Resultatet i en tabell

TRÄD	ORIGINAL MIB	DAGENS MIB	BÄSTA KLASSISKA	NVPACK MIB	MOT DAGENS
Programkod	55,3	11,5	xz −9e · 8,2	7,2	−37,2 %
Bilder (PNG)	431,0	362,9	zstd −22 · 245,2	208,9	−42,4 %
Foton (JPEG)	408,5	406,3	bzip2 −9 · 404,9	345,1	−15,1 %
Blandat projektträd	1 814,4	452,8	xz −9e · 297,5	279,1	−38,4 %

Allt är förlustfritt och bit-exakt — bevisat, inte påstått.

Varje omvandling syr ihop sitt resultat igen och jämför med originalet redan vid packningen. Stämmer inte varje byte lagras filen rå i stället. Ett arkiv kan därför aldrig bli "nästan rätt": antingen är omvandlingen bevisad, eller så användes den inte. Utöver det packas och packas varje mätning i den här rapporten upp i sin helhet, och jämförs mot originalträdet med `diff -rq`.

Tre lager som ingen vanlig packare rör

JPEG kodas om utan att en pixel rörs `src/nvjpg.c · 760 rader C`

En JPEG är DCT-koefficienter entropikodade med Huffman. Huffman kan inte koda en symbol på mindre än en bit, och vet ingenting om att blocket bredvid såg nästan likadant ut. nvjpg avkodar koefficienterna och kodar om dem med en adaptiv aritmetisk kodare vars modeller frågar tre grannskap: blocket till vänster och ovanför (*hur rörig är den här delen av bilden?*), samma plats i grannblocken (*hur stor brukar just den frekvensen vara här?*), och platserna till vänster och ovanför inuti samma block (*var ligger energin?*).

Resultat på hela fotokorpusen: **1 143 av 1 143 filer bit-exakta, −15,0 %.** Noll råfall. Modellen trimmades fram i mätbara steg: första fungerande modell −12,0 %, grannskapskontexter −13,1 %, finare hinkar −13,9 %, takt som sjunker med antalet observationer −14,6 %, blockets täthet i storleksmodellen −14,8 %. Två idéer som lät bra ströks efter mätning: egen kontextdimension för grannblockets riktning (−13,3 %, kontexterna tunnades ut) och två fasta adaptionstakter som vägs ihop (−11,6 %, den långsamma takten drog medelvärdet mot 50 % hela bilden igenom).

PNG vecklas ut och packas om `src/png.py`

Deflate från 1996 är ett dåligt skal runt bilddata: 32 kB fönster, en fil i taget. nvpack vecklar ut IDAT-strömmen till råa filterrader och packar dem tillsammans med alla andra bilders rådata. Vägen tillbaka är det svåra — deflate är inte en kanonisk kodning, så samma indata ger olika utdata beroende på nivå, minnesnivå och strategi. nvpack letar därför upp **receptet**: den uppsättning zlib-parametrar som råkar ge precis den ström filen redan har. Sökningen är billig eftersom zlib-huvudet röjer fönsterstorlek och nivågrupp, eftersom fel kandidat avviker inom de första hundra kilobytena och då avbryts, och eftersom receptet som senast fungerade provas först.

Hittas inget recept lagras filen rå, och det händer ofta: Chrome och gnome-screenshot använder deflate-motorer som systemets zlib inte kan härma. I PNG-korpusen gick 281 av 4 422 provade filer att veckla ut — vinsten på dem som gick är desto större.

En egen kompressor för källkod `src/nvcm.c · 450 rader C`

xz var det starkaste som fanns installerat, så nvpack fick något starkare. I stället för att hänvisa bakåt i en ordbok gissar nvcm nästa *bit*, med åtta modeller samtidigt: ordning 0–8, en ordmodell, och en matchmodell som gör ordbokspackarens trick som en röst bland andra. En blandare lär sig under tiden vem som brukar ha rätt, och två efterjusterare putsar resultatet.

INDATA	XZ −9E	NVCM	
källkod, 8 MB	782 876	**663 744**	−15 %
seq 1 50000	**10 160**	47 548	+368 %

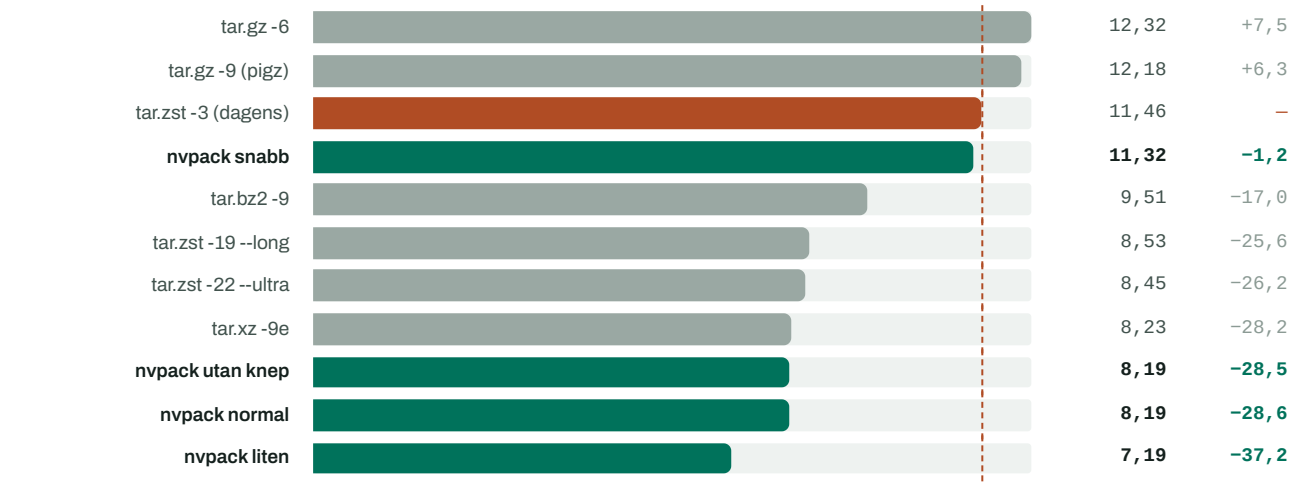
Den är alltså inte bättre på allt — en modell som gissar nästa bit ur de föregående byten ser inte att ett tal ökar med ett, medan ordbokspackarens matchningslängd gör det. Därför *provas* nvcm mot xz och zstd per block i stället för att väljas på filtyp.

Fyra träd, tolv packare

Stapeln visar hur stor del av originalet som är kvar — kortare är bättre. Det streckade märket är dagens arkivskript: allt till vänster om det är en vinst, allt till höger en förlust.

Programkod — 55,3 MiB i 5 962 filer

Källkod ur arton projekt på maskinen: .mjs, .py, .ts, .json, .html, .css, .md

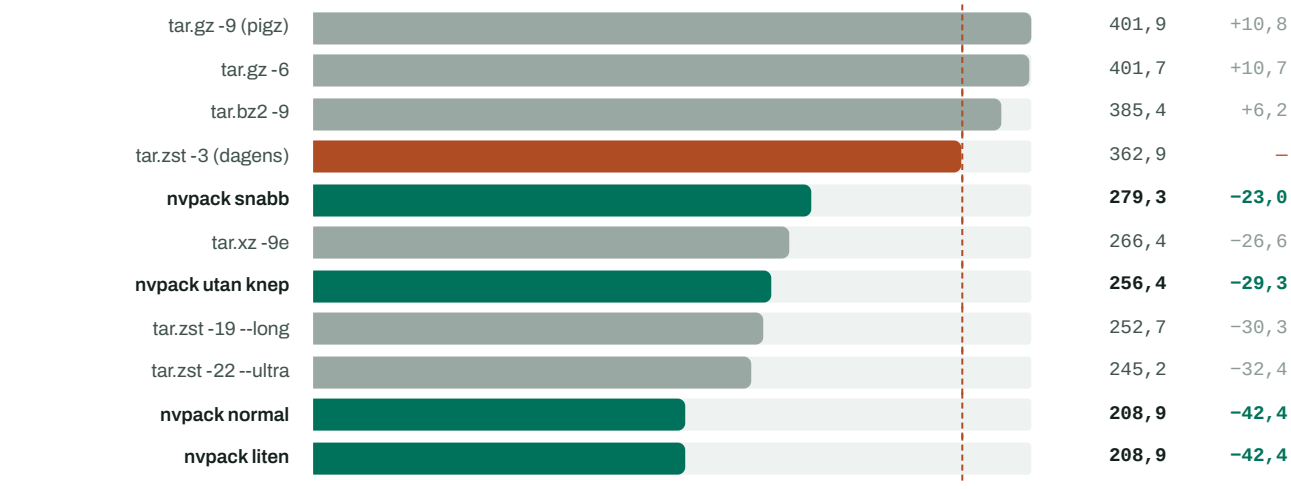


nvpack · dagens arkivskript · klassiska verktyg

stapel = andel av originalets 55,3 MiB · streck = dagens nivå

Bilder (PNG) — 431,0 MiB i 4 617 filer

Skärmdumpar, renderade produktbilder och paritetsklipp ur ting

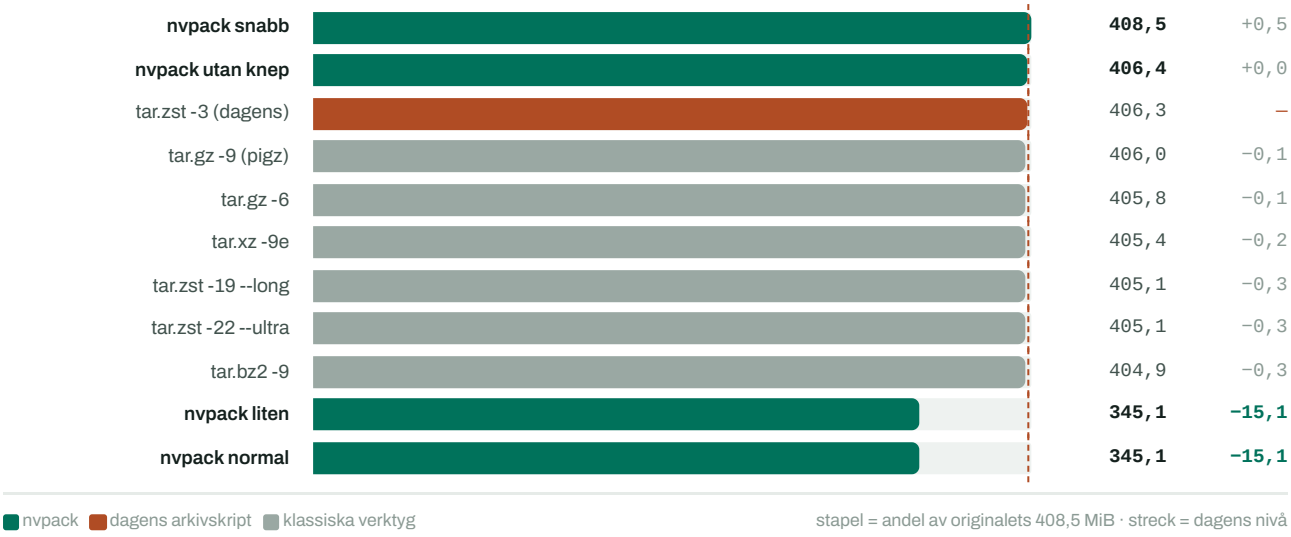


nvpack · dagens arkivskript · klassiska verktyg

stapel = andel av originalets 431,0 MiB · streck = dagens nivå

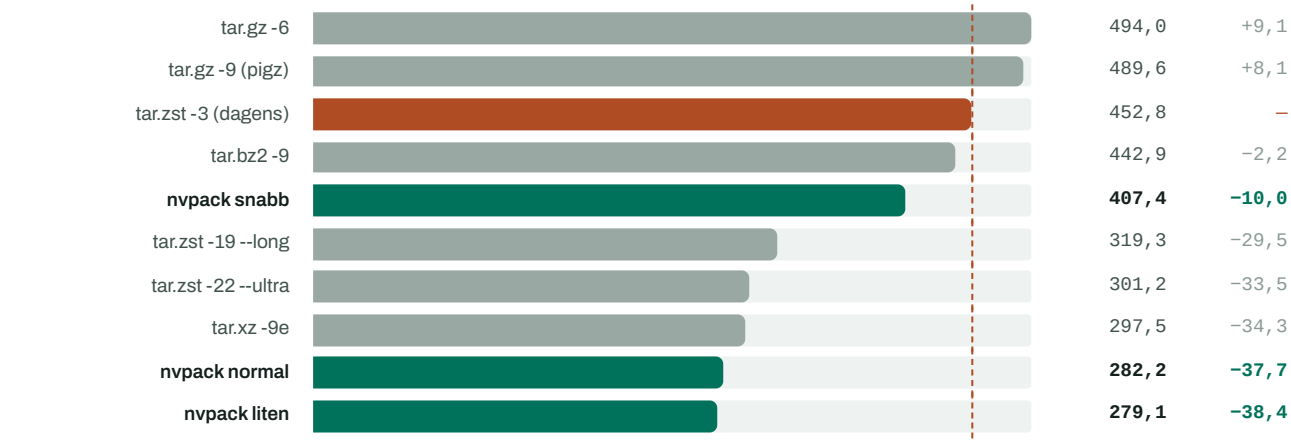
Foton (JPEG) — 408,5 MiB i 1 143 filer

Kamerabilder ur Pecka-arkivet plus TikTok-omslag — alla baseline-JPEG



Blandat projektträd — 1 814,4 MiB i 15 139 filer

Hela ~/symbios-3d: 1,2 GB delade bibliotek, 4 600 .py, 4 600 .pyc, en hel arbetskopia



Vad diagrammen säger

- **Foton är det tydligaste fallet.** Varje klassiskt verktyg — gzip, bzip2, zstd −22, xz −9e — stannar inom en procent från där det började. De kan bara leta efter upprepningar i en ström som redan är entropikodad. nvpack byter ut entropikodningen och tar 15 %.
- **På det blandade trädet vinner nvpack även på tid.** zstd −22 behövde 623 s för 301,2 MiB; nvpack tog 325 s för 279,1 MiB — halva tiden, 7 % mindre. Mot xz −9e är det 6 % mindre på i stort sett samma tid.
- **Läget `snabb` är gratislunchen.** 10 % under dagens arkiv på fjorton sekunder, helt och hållet på dedup och sortering. Där finns dubbletter som tar aldrig ser.

Tre gånger hade jag fel

Varje designbeslut i nvpack är mätt fram, och tre av dem gick tvärtemot vad som verkade självklart. De tre är anledningen till att siffrorna ser ut som de gör.

1 PNG som blåses upp hundra gånger är värda mest

Jag satte ett tak: en bild vars rådata blir mer än sexton gånger större än dess deflate-ström ansågs inte värd att veckla ut. Mätningen gav **−40 % just i den högsta klassen** — mest av alla. I efterhand är det logiskt. En bild som deflate klämt 50:1 är ett skärmklipp med stora enfärgade ytor, och när femtio sådana ligger utvecklade i samma block hittar en modern packare likheterna **mellan** bilderna. Det kunde deflate aldrig se genom sitt 32 kB-fönster, en fil i taget.

UPPBLÅSNING	FILER	ORIGINAL	ZSTD19 PÅ ORIGINAL	VIA UTVECKLING	VINST
2–4×	13	17,5 MB	17,48 MB	13,92 MB	20,4 %
4–8×	14	32,1 MB	26,51 MB	18,45 MB	30,4 %
8–16×	40	61,5 MB	49,93 MB	40,95 MB	18,0 %
16–32×	26	17,7 MB	12,24 MB	9,88 MB	19,3 %
32×+	188	13,4 MB	7,31 MB	4,35 MB	40,4 %

2 "Redan packat" får inte vara ett antagande

Film, zip och PNG som inte kunde vecklas ut lagrades rått per definition — det sparar packtid på data där ingen komprimerare ändå hittar något. På PNG-korpusen kostade det **175 MiB**, och på det skarpa lostconnection-arkivet gjorde det nvpack **sämre än dagens**: en halv procent av 2,9 GB film är fjorton megabyte, och `zstd -3` tar dem på några sekunder. Nu provpackas varje block på åtta ställen med varje kodek, och tröskeln har tre utfall i stället för två — rått under 0,2 %, billig kodek däremellan, dyr kodek när det finns något att hämta.

3 Provsviten hittade tyst dataförlust

En symlänk som pekar på en katalog **försvann helt ur arkivet** — utan fel, utan varning. Den dyker upp bland de *kataloger* `os.walk` rapporterar, inte bland dess filer, och koden letade bara efter länkar bland filerna. Katalogernas egna tidsstämplar återställdes inte heller. Båda hade gått obemärkt förbi i **varje storleksmätning** — mätningar jämför innehåll, inte träd. Det är därför provsviten bygger ett träd av kantfall: tomma filer, trasiga länkar, filnamn med nyrad i, udda rättigheter, tidsstämplar från 1985 och 2031.

Skillnaden är ett lager, inte en inställning

gzip, bzip2, xz och zstd är bytestrømspackare: de ser en enda ström och vet ingenting om filer eller filtyper. tar är en behållare utan komprimering, utan index och utan dedup. nvpack bor i tomrummet mellan dem.

	SER FILER	INDEX	DEDUP	OMVANDLAR	VÄLJER KODEK
gzip / bzip2 / xz / zstd	nej	nej	via fönstret	nej	nej
tar	ja	nej	nej	nej	—
zip	ja	ja	nej	nej	nej
7-Zip	ja	ja	nej	delvis	per typ
restic / borg	ja	ja	starkare	nej	nej
nvpack	ja	ja	ja	ja	mätt, per block

Vad som är nytt — och vad som inte är det

Ingen av de tre omvandlingarna är uppfunnen här. PNG-receptsökningen gör `precomp` och `reflate` sedan länge. Förlustfri JPEG-ompackning gör `lepton` och `brunsli` — och *bättre* än min, cirka 22 % mot mina 15 %. Kontextblandning är paq- och zpaq-familjen, och de är långt starkare än nvcm. Skillnaden är inte uppfinningen utan **integrationen**: ett enda arkivprogram som gör allt det här, och som för varje omvandling bevisar att den går att vända innan den används. Inget av de klassiska verktygen försöker ens; de specialiserade gör det var och en för sin filtyp, och ingen av dem fanns installerad på den här maskinen.

Vad nvpack inte kan som de andra kan

- **Inte strömmande.** tar kan rapa rakt in i ssh eller på band. nvpack behöver en katalog och arbetsyta på disk — upp till flera GB när PNG-utvecklingen är igång.
- **Oberoende block kostar korsredundans.** Därför förlorar nvpack mot `tar.zst -19` på PNG-korpusen när dedup och omvandlingar är avstängda: 256,4 mot 252,7 MiB. zstd ser en enda 431 MB-ström genom ett 2 GB-fönster; nvpack klipper vid 192 MB och packar varje bit för sig. Oberoendet köper parallellitet och delvis uppackning, och det kostar det här.
- **Inget standardformat.** En `tar.gz` går att läsa år 2050. En `.nv` går att läsa så länge `~/nvpack` finns.
- **Ingen kryptering, ingen felkorrigering, inga inkrementella snapshots.** Där är restic rätt verktyg.
- **Hårda länkar blir fristående kopior**, och rör, uttag och enheter hoppas över med varning. **Progressiv JPEG** och **PNG ur okända deflate-motorer** lagras rå.
- **nvcm är långsam i båda riktningarna** — 207 s att packa upp det blandade trädet, mot 8 s för läget `normal`.

Och så verkligheten: ditt eget arkiv

Korpusarna i den här rapporten är rena. Ett riktigt projektträd är blandat, och blandningen avgör vad man faktiskt tjänar. På hela `~/lostconnection` (2,9 GB, 2 009 filer) är vinsten **−0,09 %**: 2 869,25 → 2 866,74 MiB. Utan `data/klipp` (25 MB, 432 filer) är den **−3,8 %**: 17,17 → 16,52 MiB.

Det är det ärliga svaret för just det arkivet — **ingenting hjälper, för det ÄR 517 filmer.** Film är redan packad av en videokodare, och där finns inget motsvarande entropilager att byta ut. Byt alltså inte ut `scripts/arkivera.sh` för storlekens skull.

BILAGA

Alla mätvärden

Varje rad är en hel rundgång: packa, packa upp, och `diff -rq` mot originalträdet. Kolumnen längst till höger är utfallet av den jämförelsen, inte en förhoppning. Korpusarna byggs av `korpus/bygg-korpus.py` ur riktiga projekt på maskinen, som hårdlänkar — noll extra diskutrymme, exakt samma bytes.

Programkod — 55,3 MiB i 5 962 filer

PACKARE	MIB	KVOT	MOT DAGENS	PACKNING	UPPACKNING	BIT-EXAKT
tar (okomprimerad)	60,83	0,91×	+430,8 %	0,1 s	0,1 s	ja
tar.gz -6	12,32	4,49×	+7,5 %	2,2 s	0,3 s	ja
tar.gz -9 (pigz)	12,18	4,54×	+6,3 %	0,5 s	0,3 s	ja
tar.bz2 -9	9,51	5,81×	−17,0 %	7,5 s	2,3 s	ja
tar.zst -3 (dagens)	11,46	4,82×	—	0,2 s	0,1 s	ja
tar.zst -19 --long	8,53	6,48×	−25,6 %	34,7 s	0,1 s	ja
tar.zst -22 --ultra	8,45	6,54×	−26,2 %	56,4 s	0,1 s	ja
tar.xz -9e	8,23	6,72×	−28,2 %	46,8 s	0,6 s	ja
nvpack snabb	11,32	4,88×	−1,2 %	2,7 s	0,8 s	ja
nvpack utan knep	8,19	6,75×	−28,5 %	42,6 s	1,2 s	ja
nvpack normal	8,19	6,75×	−28,6 %	43,2 s	1,2 s	ja
nvpack liten	7,19	7,69×	−37,2 %	71,3 s	67,9 s	ja

Bilder (PNG) — 431,0 MiB i 4 617 filer

PACKARE	MIB	KVOT	MOT DAGENS	PACKNING	UPPACKNING	BIT-EXAKT
tar (okomprimerad)	438,41	0,98×	+20,8 %	2,8 s	0,3 s	ja
tar.gz -6	401,67	1,07×	+10,7 %	20,1 s	2,5 s	ja
tar.gz -9 (pigz)	401,89	1,07×	+10,8 %	1,8 s	2,6 s	ja
tar.bz2 -9	385,38	1,12×	+6,2 %	73,9 s	44,0 s	ja
tar.zst -3 (dagens)	362,85	1,19×	—	0,6 s	0,3 s	ja
tar.zst -19 --long	252,73	1,71×	−30,3 %	39,8 s	0,5 s	ja
tar.zst -22 --ultra	245,22	1,76×	−32,4 %	156,2 s	0,8 s	ja
tar.xz -9e	266,41	1,62×	−26,6 %	128,4 s	4,9 s	ja
nvpack snabb	279,28	1,54×	−23,0 %	2,8 s	0,8 s	ja
nvpack utan knep	256,42	1,68×	−29,3 %	44,6 s	1,7 s	ja
nvpack normal	208,88	2,06×	−42,4 %	254,0 s	9,5 s	ja
nvpack liten	208,88	2,06×	−42,4 %	250,8 s	9,3 s	ja

Foton (JPEG) — 408,5 MiB i 1 143 filer

PACKARE	MIB	KVOT	MOT DAGENS	PACKNING	UPPACKNING	BIT-EXAKT
tar (okomprimerad)	409,37	1,00×	+0,8 %	2,1 s	0,2 s	ja
tar.gz -6	405,81	1,01×	−0,1 %	19,1 s	2,0 s	ja
tar.gz -9 (pigz)	406,04	1,01×	−0,1 %	1,3 s	1,6 s	ja
tar.bz2 -9	404,94	1,01×	−0,3 %	56,8 s	40,9 s	ja
tar.zst -3 (dagens)	406,30	1,01×	—	0,5 s	0,2 s	ja
tar.zst -19 --long	405,09	1,01×	−0,3 %	53,1 s	0,4 s	ja
tar.zst -22 --ultra	405,09	1,01×	−0,3 %	225,4 s	0,4 s	ja
tar.xz -9e	405,43	1,01×	−0,2 %	157,3 s	3,5 s	ja
nvpack snabb	408,50	1,00×	+0,5 %	1,1 s	0,4 s	ja
nvpack utan knep	406,35	1,01×	+0,0 %	12,6 s	0,6 s	ja
nvpack normal	345,07	1,18×	−15,1 %	21,5 s	7,8 s	ja
nvpack liten	345,07	1,18×	−15,1 %	27,9 s	7,9 s	ja

Blandat projektträd — 1 814,4 MiB i 15 139 filer

PACKARE	MIB	KVOT	MOT DAGENS	PACKNING	UPPACKNING	BIT-EXAKT
tar (okomprimerad)	1 804,14	1,01×	+298,4 %	6,6 s	1,5 s	ja
tar.gz -6	494,04	3,67×	+9,1 %	98,9 s	10,0 s	ja
tar.gz -9 (pigz)	489,62	3,71×	+8,1 %	19,1 s	9,4 s	ja
tar.bz2 -9	442,88	4,10×	−2,2 %	207,6 s	82,1 s	ja
tar.zst -3 (dagens)	452,83	4,01×	—	2,1 s	3,1 s	ja
tar.zst -19 --long	319,30	5,68×	−29,5 %	148,9 s	4,3 s	ja
tar.zst -22 --ultra	301,21	6,02×	−33,5 %	622,9 s	4,1 s	ja
tar.xz -9e	297,54	6,10×	−34,3 %	289,3 s	5,3 s	ja
nvpack snabb	407,37	4,45×	−10,0 %	14,2 s	3,8 s	ja
nvpack normal	282,25	6,43×	−37,7 %	315,4 s	8,3 s	ja
nvpack liten	279,10	6,50×	−38,4 %	325,4 s	207,4 s	ja

Så kör du det

```
cd ~/nvpack && make                        # bygger C-delarna
bash prov/prova-allt.sh                    # provsviten, alla kantfall

./src/nvpack packa ~/projekt ark.nv --läge normal
./src/nvpack packa-upp ark.nv /tmp/åter
./src/nvpack lista ark.nv                  # block, kodek och kvot
./src/nvpack prova ~/projekt               # packa, packa upp, jämför

python3 matning/matning.py kod             # mät om en korpus
python3 matning/rapport.py --in-i-readme   # bygg om tabellerna
python3 rapport/bygg-tryck.py              # bygg om den här PDF:en
```

Tre lägen: `snabb` motsvarar dagens arkivskript (zstd −3, inga omvandlingar), `normal` är standard, och `liten` släpper dessutom in nvcm på de textblock där provet visar att den vinner.