Looped ternary encoders for typed decisions: a recipe from laptop to browser

Maxime Golfier

https://github.com/maxgfr/skycmd

DOI 10.5281/zenodo.24420817

21 November 2026

Abstract

How to build a typed-decision model of 251,104 bytes, train it on one laptop and run the same file in a browser. The graph encodes the question and its options once and caches them, gives each measured value one token (9 instead of 16 for a Flappy frame), loops shared blocks with an exit after each pass, and stores ternary weights at 2 bits. Where a choice was measured, we say what it bought: training the ternary weights in recovers the 6 to 11 points of validation accuracy that rounding a float model loses; three DAgger rounds take s3-pico from 145.3 Flappy pipes with 14 crashes in 30 episodes to 200 of 200 with none; exit thresholds chosen by playing keep 200 of 200 at 1.44 of 4 passes, where thresholds fitted on held-out questions crash s2-nano in every Flappy episode; the browser's probabilities stay within 9.5e-7 of the native engine's; and on these small matrices float kernels beat packed ternary ones (3.99 against 12.23 µs per product), so 2 bits is a storage format. The same blocks give a 631,936-byte end-of-turn detector (2.39 ms per decision in WebAssembly) and a tool router that fuses a trained model with a static embedding without further training. Every piece is adopted from prior work, each recipe was trained once on one seed, and Table 3 lists what was never measured. A companion report [1] holds the protocol and every gate as read.

Technical report, not peer-reviewed. This report has not been peer-reviewed, nor reviewed by anyone outside the project. Every number in it is copied from a page or a file of the repository, and each one links to its evidence. Where a number is computed here, the text says so. It describes the methods; the protocol and every gate's reading are in the companion report [1]. The web version of this report is at https://maxgfr.github.io/skycmd/paper/methods/.

1 Introduction

A *decision model* receives a state (a game frame, a short text, the last seconds of audio) and one or more typed questions, and returns a probability for every allowed option in one forward pass, without generating text. A `choice` question picks one of several named options, a `score` question one of several ordered levels, and a `noul` question is a probabilistic yes or no (`docs/SPEC.md`). TypeSafe introduced models that answer with type-safe structured values and calibrated probabilities, its System One Models, with Jev, opened in early access on 15 September 2026 [2]; the three type names are those of the project's specification. Open models of this kind now range from 706,048 parameters for one form-filling task [3] to 421M [4].

skycmd trained such models from scratch under three constraints: small files, from 142,912 bytes to 3,692,288 bytes for the s6 family and most of them under one megabyte (manifest); one Apple M5 laptop with 16 GB of memory, the machine its owner works on, for every run (`docs/big-run-report.md`); and the browser, where every demo runs the file its numbers were measured on.

The companion report [1] is the record of the protocol and of every gate's reading. This report is its counterpart on method: it describes what the code does, gives at most one sentence per system on its result, with a pointer, and follows the code where the project's specifications disagree with it.

It contributes five descriptions, each with the parts it adopts named as such, and the lessons they leave.

1. **The decision graph** (section 3): two streams and a cache, options isolated by the attention mask, one token per measured value, shared blocks looped with an exit after each pass, a fusion head shared by the exits, and ternary

weights trained in. Each piece is adopted: order invariance from Set-LLM [5] and Amit [6], a looped shared block with an answer after each loop from TRM [7] and Ouro [8], one token per value from FT-Transformer [9] and FoNE [10], ternary weights from BitNet b1.58 [11]. Built from them, s3-pico is, among the open decision models we found that take their options at request time (searches made in November 2026, the last on 21 November), the smallest file that accepts all three question types, and the only one with published ternary weights; the companion report names systems that are smaller or ternary under other conditions (What we bring, section 1).

2. **A training recipe for one laptop** (section 4): masked-language pretraining with a loss on numbers, a loss at every exit, soft labels from an open-weight teacher kept only where they agree with the gold answer, DAgger [12] on the games, and budgets counted in working minutes. A recipe table gives the settings of seven published files.
3. **Exit thresholds chosen by playing** (section 5): temperatures per exit, type and bucket, and a threshold accepted only if the games played at that threshold stay within 2% of full depth. DeeR-VLA [13] sets the exit thresholds of a robot policy under compute and memory budgets and, in an online variant its body describes, against task success; the readings here are consistent with that line of work and do not introduce it. A conformal variant is described with how the rule judged it on each model.
4. **Task-specific systems on the same blocks** (section 6): an end-of-turn student of a larger audio model, a tool router fused with a static embedding, a verifier of secret candidates, and a forecaster. The verifier's design, rules that propose and a learned model that judges, is that of Samsung's CredSweeper [14] and GitGuardian's FP Remover [15].
5. **One inference engine in Rust** (section 7): the GGUF files, the ternary kernels, the caches and the WebAssembly builds, with the measured agreement between the browser and the native engine.
6. **Lessons for practitioners** (section 8): what each choice bought where it was measured, how strong that evidence is, and what was never measured.

What the combination adds is a chain from data to a browser in which every step fits one laptop and every claim is measured; no single piece is new (What we bring).

To use a file, start at Use it for real: two files are usable with a safety net (s6-ear2-nano with a fallback timer; s6-route when tool names are distinctive, otherwise BM25).

2 Related work

Each reference was opened on its arXiv, publisher or project page before it was cited; the project's Sources page lists them all.

Decision models. TypeSafe's Jev [2] came first; Laya [4] and oscar-1-17m [16] are open models of this kind on pretrained encoders, and two smaller ones appeared in 2026: cua-s1-forms [3], a byte-level encoder of width 128 for one form-filling task, and TinyDecide [17], fine-tuned from ELECTRA-small. The models of this report are trained from scratch.

Options, depth and exits. Set-LLM [5] and Amit [6] isolate the options from one another, and Jeva [18], concurrent work, also reuses their representations across requests. Weight sharing across depth goes back to Universal Transformers [19], and learning how many steps to compute to ACT [20] and PonderNet [21]; LoopFormer [22] trains a looped Transformer on trajectories of several lengths so that short loops stay informative, the aim of the loss at every exit here. A loss at every exit follows deeply-supervised nets [23], and skipping loops at random in training follows stochastic depth [24]. Early exits go back to BranchyNet [25], DeeBERT [26] and FastBERT [27]; Meronen et al. [28] correct the overconfidence of dynamic networks with a probabilistic treatment of the last layers, where this report fits a temperature per exit (section 5.1). CATs [29] calibrate a stopping rule by a conformal method so that early answers agree with the full model, and conformal risk control [30] gives exit thresholds a guarantee, in the setting of CALM [31] and Fast yet Safe [32]; SAFE-KD [33] calibrates the stopping thresholds of each exit by conformal risk control. DeeR-VLA [13] sets the exit thresholds of a robot policy under compute and memory budgets, from demonstrations or, in an online variant its body describes, by Bayesian optimisation against task success.

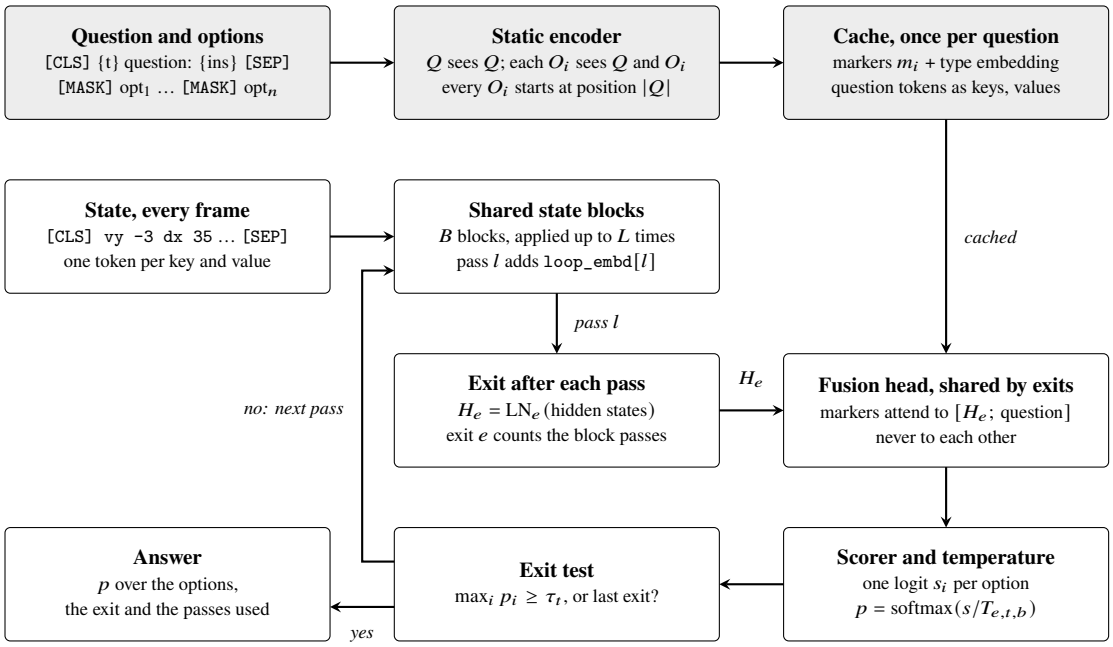
Weights and training. The straight-through estimator [34] trains the ternary weights. Ternary weights on small networks are studied in BitNet b1.58 Reloaded [35], and scaling binary or ternary weights by a mean of absolute values goes back to XNOR-Net [36] and Ternary Weight Networks [37]. The encoder block keeps much of the layout of ModernBERT [38], with rotary position embeddings [39] (section 3.4 says what differs). Temperature scaling and the expected calibration error with 15 equal-width bins are those of Guo et al. [40]; learning a teacher's soft labels is

knowledge distillation [41]. The s6 models are trained with AdamW [42]. The code includes the Muon optimizer [43]; that its gains over AdamW shrink once AdamW is tuned as carefully [44], and that AdamW can win at equal time on small models [45], is what the project's ablation found too (section 4.4).

Task-specific work. The closest small end-of-turn work found is Helwani et al. [46], whose proposed model, a compact MFCC-based student distilled from wav2vec 2.0 representations, has 1.14M parameters. Routing by embedding similarity is the method of semantic-router [47], and the router's lexical score is BM25 [48]. The verifier is trained on Samsung's CredData [49]. Fracast-0 [50] is a smaller forecaster that, by its abstract, does better than the one here. Training in Rust and running in WebAssembly is also done by frameworks such as Burn [51]; the teachers and baselines of each system are cited in section 6.

3 The decision model

Figure 1 shows the graph, `skycmd-s2` in the files, used by s3-pico, s6-nano, s6-micro, the router s6-route-nano and the verifier s6-secrets-verify-pico. The training graph is `crates/sky-train/src/s2/model.rs` and the inference graph `crates/sky-infer/src/s2/model.rs`; the specifications are `docs/SPEC-S2.md`, `docs/SPEC-S3.md` and `docs/SPEC-S6.md`.



Shaded: computed once per question and cached. White: every frame.
Linear layers ternary, $w \in \{-1, 0, +1\} \cdot \gamma$ (2 bits each); embeddings int8; norms f32.

Figure 1. The decision graph. The question and its options are encoded once and cached (shaded); each frame, the state goes through B shared blocks up to L times, and after each pass an exit feeds the fusion head, which is shared by all exits. The answer is returned at the first exit whose top probability reaches the threshold of the question's type, or at the last. The figure is written by `scripts/paper-methods-figure.py`; its text alternative is in the SVG.

3.1 Requests and tokenization

In short. A byte-level BPE of 1,024 to 4,096 entries, trained on the text the model will read, keeps the embedding table close to the size of the network; its size was not ablated.

A request holds a state and one or more questions; a question has a type, an instruction and its options. The options are rendered as Laya's runtime and llama.cpp's server render them: a `choice` option is `key` or `key: description`, a `score` option is `level i: description` for i = 0, 1, and so on, and the two `noul` options are `false: …` and

`true: …`, with the defaults `no, the statement does not hold` and `yes, the statement holds` (`crates/sky-infer/src/prompt.rs`).

The tokenizers are byte-level BPE in the GPT-2 style with an NFC normaliser, trained with Hugging Face `tokenizers` at 1,024, 2,048 or 4,096 entries on the text the model will read: each question head, each option with its leading space, and each state (tutorial, chapter 07). Their special tokens have fixed ids: `[PAD]` 0, `[UNK]` 1, `[CLS]` 2, `[SEP]` 3, `[MASK]` 4 and `[NUM]` 5 (`docs/SPEC-S2.md` §2). A small vocabulary keeps the embedding table close to the size of the network; the game generators clamp their values to small ranges for the same reason.

3.2 One token per measured value

In short. Each measured value rides on one token, which takes a Flappy frame from 16 tokens to 9; what the value channel adds to accuracy was not measured apart (section 8).

Before BPE, every match of `-?\d+(?:\.\d+)?` on ASCII digits is replaced by `[NUM]` and its value is kept in a parallel array, unless the match is glued to a preceding ASCII letter, as in `b2` or `glm-5` (`crates/sky-infer/src/num.rs`). With `fuse_numbers`, set in every published s3 and s6 decision file, a `[NUM]` is removed and its value moved onto the token before it, unless that token is `[CLS]`, `[SEP]` or `[MASK]` or already carries a value; then the `[NUM]` stays a token of its own. A pair such as `vy -3` is then one token. The merge happens before truncation, so every window counts merged tokens (`crates/sky-train/src/s2/data.rs`). The embedding of token i, whose id is id_i, whose value is v_i and whose flag χ_i is 1 when it carries a value, is

$$\begin{aligned} u(v) &= \operatorname{sign}(v)\,\ln(1+|v|), \qquad \omega_k = 2^k\pi/8, \quad k = 0,\dots,7, \\ \varphi(v) &= \big[\, u,\ \sin(\omega_0 u),\dots,\sin(\omega_7 u),\ \cos(\omega_0 u),\dots,\cos(\omega_7 u) \,\big] \in \mathbb{R}^{17}, \\ h_i^{(0)} &= W_{\mathrm{tok}}[\mathrm{id}_i] + \chi_i \cdot \big(W_{\mathrm{num}}\,\varphi(v_i) + b_{\mathrm{num}}\big), \qquad W_{\mathrm{num}} \in \mathbb{R}^{d\times 17}. \end{aligned} \tag{1}$$

The signed logarithm compresses magnitudes, and the eight frequencies give periods from 16 down to 1/8 in u (computed here from ω_k), so nearby values get nearby embeddings and orders of magnitude stay apart; the engine computes the sixteen sines and cosines from one pair by double-angle steps. The features are the periodic embeddings of Gorishniy et al. [52] with fixed frequencies instead of learned ones, on the signed logarithm; xVal [53] also gives a number one token, by scaling a shared embedding with the value, and its regression head on masked values is the model of the value loss of section 4.2. On the Flappy state used by the benchmarks, value fusion gives 9 tokens instead of 16, `[CLS]` and `[SEP]` included (`BENCH.md`, section "Inference").

3.3 Two streams, a cache and isolated options

In short. The question and its options are encoded once and cached, so a new frame or a new request about the same options encodes only the state (section 7 gives what the cache saves), and since no option sees another, the answer cannot depend on the options' order.

The **static stream** holds the question and its options. Its tokens are Q = `[CLS] {t} question: {ins} [SEP]`, followed for each option i by O_i = `[MASK] {option}`, with no final separator. The attention mask lets Q attend to Q and each O_i attend to Q and to itself only. Positions for the rotary embedding are 0 to |Q| − 1 for Q, and every option starts again at |Q|, so each option's `[MASK]` marker sits at the same position. Long options are cut as Appendix A.2 says. The static encoder's output at each marker is the option's marker vector m_i, and its outputs at all |Q| question positions, `[CLS]` and `[SEP]` included, are the question tokens.

Since no option attends to another and all sit at the same positions, permuting the options permutes the marker vectors and, through the head of section 3.5, the probabilities. This holds by construction, as in the prior work of section 2; on the s2 models, permuting the options moved no probability by more than 3e-6 on the CPU (`BENCH.md`, section "s2 training runs").

The **state stream** is `[CLS]` state `[SEP]`, at most max_state tokens, at positions 0 to S − 1, with full bidirectional attention. It does not see the question, so one encoding of a state serves every question asked about it, such as the action and danger questions of one Flappy frame.

The engine keeps an LRU **cache** of 16 static streams, keyed by type, instruction and options. An entry holds the marker vectors, the keys and values of the question tokens for every fusion block, and the first fusion block's queries with their attention over the question tokens already computed. A second cache of 4 entries, keyed by the state text, keeps the

exits already computed for a state (`crates/sky-infer/src/s2/engine.rs`). In a game, the same questions come back every frame, so only the state is new.

3.4 Shared blocks, loops and exits

In short. A few shared blocks looped with an exit after each pass give depth without more parameters and an answer after every pass, so an easy decision can stop early: s3-pico answers Flappy in 1.44 of its 4 passes at the threshold chosen by playing (section 5.3); a looped block was never compared with a stack of distinct blocks of the same size.

Both encoders are built from one kind of block. It applies a LayerNorm without bias (ε = 1e-5), a fused query, key and value projection without bias, rotary embeddings [39] in the rotate-half form with base 10,000 on queries and keys, scaled dot-product attention and an output projection without bias; then a second LayerNorm and a GeGLU feed-forward: W_{up} LN(x), with W_{up} of shape 2ff × d, is split into a gate half and a value half, and x ← x + W_{down} (gelu(gate) ⊙ value). As in ModernBERT [38], the first block of each encoder has no attention norm on its first application; unlike ModernBERT, there is no norm after the embeddings. The number of heads is set per file (1 for s3-pico, 2 for s6-nano, 4 for s6-micro; Table 1), or d / 64 by default.

The state encoder has B distinct blocks applied in order, and the sequence is repeated L times. From the second loop on, block 0 normalises its input with a norm of its own, and before loop $l \geq 1$ (counted from 0) a learned loop embedding loop_embd[l] is added to every hidden state, so the shared block knows which pass it is in. Exits sit after given numbers of block passes; exit e normalises the hidden states with its own LayerNorm and gives H_e. s3-pico has B = 1, L = 4 and exits after passes 1 to 4; s6-nano B = 2, L = 2 and exits after 1 to 4; s6-micro B = 2, L = 3 and exits after 2, 4 and 6 (`configs/`). The static encoder is a plain stack of 1 or 2 blocks with a final norm.

3.5 Fusion head, scorer and probabilities

In short. One fusion head, shared by every exit, scores each option against the state and the question without letting the options see each other, and a temperature per exit, type and bucket of option counts calibrates its probabilities.

The fusion head turns each option's marker into a score, once per exit, with the same weights for every exit. Each option starts from $x_i = m_i$ + type_embd[t], its marker plus an embedding of the question's type. Each of the 2 fusion blocks applies a cross-attention whose queries are the options and whose keys and values are [H_e; question tokens], the state after exit e followed by the cached question tokens; then a ReLU feed-forward of width 2d. Every LayerNorm and linear layer of the head has a bias, and there is no rotary embedding. The options attend to the state and the question, never to each other. The scorer gives one logit per option and the probabilities follow:

$$s_i = w_{\text{o}} \cdot \text{gelu}\big(W_{\text{c}}\,\text{LN}(x_i) + b_{\text{c}}\big) + b_{\text{o}}, \qquad p_i^{(e)} = \frac{\exp(s_i/T)}{\sum_j \exp(s_j/T)}, \qquad T = T_{e,\,t,\,b(n)}. \tag{2}$$

The temperature depends on the exit e, the type t and the bucket $b(n)$ of the number of options n: 2 or fewer, 3 to 5, 6 to 10, or 11 and more (section 5.1). Appendix A.2 gives the order in which the engine looks a temperature up. The answer depends on the type (`crates/sky-infer/src/engine.rs`):

$$\begin{gathered}\text{choice: } \arg\max_i p_i, \text{ with confidence } \max\big(0,\ (p_{\max} - 1/n)/(1 - 1/n)\big);\\ \text{score: the expected level } \textstyle\sum_i i \cdot p_i \text{ over the levels } i = 0, \ldots, n-1; \qquad \text{noul: } p_1, \text{ the probability of } \texttt{true}.\end{gathered} \tag{3}$$

At inference the exits run in order, and the engine answers at

$$e^* = \min\Big(\big\{\, e : \max_i p_i^{(e)} \geq \tau_t \,\big\} \cup \{N_x\}\Big), \tag{4}$$

where N_x is the last exit and τ_t is one threshold per question type, shared by all exits; the test uses the calibrated probabilities (`crates/sky-infer/src/s2/engine.rs`). A file without a threshold reads 0.9. A threshold of 1 makes the engine start directly at the last exit, so the earlier exits are not computed; the s6 files that the manifest marks as published at full depth store 1 for each type (section 5.4). The answer reports the exit and the number of block passes used.

3.6 Ternary weights, trained in

In short. Ternary weights trained in, rather than rounded after training, recover the 6 to 11 points of validation accuracy that rounding cost the s2 models, and store s3-pico's 445,345 parameters in 251,104 bytes.

Every linear layer of the encoders and of the fusion head is ternary, and so is the scorer's hidden layer. For a weight matrix W, output row o is quantized with its own scale:

$$\gamma_o = \max\Big(\tfrac{1}{n_{\text{in}}} \textstyle\sum_j |W_{oj}|,\ 10^{-8}\Big), \qquad \hat{W}_{oj} = \gamma_o \cdot \text{clip}\big(\text{round}(W_{oj}/\gamma_o), -1, 1\big), \qquad W_{\text{used}} = W + \text{sg}(\hat{W} - W), \tag{5}$$

where n_{in} is the number of inputs (columns) and sg stops the gradient: the forward pass uses $\hat{W}$, and the gradient reaches the float master weights W unchanged, the straight-through estimator [34] (`crates/sky-train/src/s2/model.rs`). The scale is the mean absolute value (absmean) of BitNet b1.58 [11], which takes it over the whole tensor; taking it per output row is the per-filter scale of XNOR-Net [36], and Ternary Weight Networks [37] also scale ternary weights by a mean of absolute values. The scorer's output layer, the value projection W_{num}, the type and loop embeddings, the norms and the biases stay in float. The token embedding is trained in float and stored in int8 at export, with one scale per row, max |row| / 127; it is not simulated in training (`crates/sky-train/src/s2/gguf.rs`).

Quantization starts when the training progress, measured in working time over the budget or in steps, reaches 1 − q, where q is the fraction of the run in QAT. The scripts pass q = 0.9, a float warm-up of 10%, for a model's first decision training, and q = 1.0, every step ternary, for the DAgger rounds; the command-line default, q = 0.3, is the one of the first specification (`scripts/train-s6.sh`). At the switch the trainer validates both forms, saves the float weights, and leaves the learning-rate schedule as it is. Why train the ternary weights rather than round a float model: on the s2 models, ternarizing without QAT cost 6 to 11 points of validation accuracy (pico 0.470 to 0.406, nano 0.542 to 0.432), and 30% of the budget in QAT recovered it (pico 0.482, nano 0.556) (`BENCH.md`, section "s2 training runs").

In the file, each ternary matrix X becomes `X.tq`, bytes of type I8 holding 4 weights each, 2 bits per weight from the least significant bits up (00 for 0, 01 for +1, 10 for −1; 11 is rejected), each row padded to a multiple of 4 weights, and `X.scale`, one 16-bit float per row. Everything else is stored in 32-bit float. This is how s3-pico's 445,345 parameters take 251,104 bytes, against 1.8 MB in 32-bit float (`BENCH.md`, section "skycmd-s3").

4 Training

4.1 Data

In short. Games labelled by exact oracles, public decision sets, and an open-weight teacher's soft labels kept only where they agree with the gold answer; that filter kept 560,238 of 747,077 labels, and its effect was not measured.

Game oracles. Four games are written in Rust: Flappy Drone, Rescue Run, Obstacle Dodge and a Doom-like game (`crates/sky-games/`). Each writes its state as short `key value` pairs of small binned integers and computes the useful relations in code: Flappy gives the room above and below the drone rather than absolute heights, so the model maps measurements to a decision (tutorial, chapter 20). An exact oracle gives a target distribution over the options at every decision. The generator draws an exploration rate per episode from 0, 0.05, 0.15 and 0.3, caps Flappy episodes at 12 pipes (DAgger collection included), sends 30% of the records to each game's secondary question, keeps records with a probability that balances the classes, and splits by episode seed. It renders 30% of the records in French, shuffles the options of a `choice` question, replaces an option's key by one of its synonyms with probability 0.5 where it has some, and drops 15% of the option descriptions (`crates/sky-games/src/gen.rs`).

Public sets and a numeric curriculum. Hugging Face decision sets are converted into one record format, with a type, an instruction, rendered options, a state, a target distribution and a split; test-only sets such as `typed_decisions_bench` never reach training (tutorial, chapter 05). A numeric curriculum of `key value` states with exact answers (which key is largest, is a greater than b, and so on) is mixed in at a weight of 0.1 (`docs/SPEC-S3.md` §3). The s6 models are pretrained on the non-web parts of Dolma 3 [54] and TxT360 [55], then on the earlier web text (tutorial, chapter 27; `DATA_LICENSES.md`).

Teacher labels. The teacher of s6-nano and s6-micro is GLM-5.3-flash [56], an open-weight model under the MIT licence, called through its provider's API, so the laptop does not run it. It receives batches of 20 training questions, each with its type, instruction, options and state cut to 2,000 characters, and is asked for one probability per option, summing to 1, at a sampling temperature of 0.2. Malformed labels are dropped and the others divided by their sum.

Only training questions with 2 to 12 options are sent, never a benchmark, held-out or teacher-generated source, and a batch refused by the provider's moderation is halved until each refused text stands alone (`crates/sky-data/src/distill.rs`, `docs/SPEC-S6.md` §15). A label is kept only when its argmax is the gold answer, and the target becomes $0.5 \cdot \text{gold} + 0.5 \cdot \text{teacher}$, in a separate `distill` group next to the gold records. The teacher labelled 747,077 of the 747,198 selected questions, the moderation refusing the other 121, and 560,238 of the 747,077 labels agree with the gold answer and are kept (`docs/big-run-report.md` §7). Smart Turn v3.2 and TinyCast, the other teachers, come with their systems in section 6.

4.2 Pretraining

In short. Masked-language pretraining with a small loss on the masked values, weighted 0.05 rather than the specification's 0.5, because at 0.5 the number error kept the token loss on its unigram plateau.

The state stack is pretrained as a masked language model (`crates/sky-train/src/s2/model.rs`). 30% of the tokens are masked, at least one per row: 80% become `[MASK]`, 10% a random token and 10% stay. A masked `[NUM]` keeps its id, its value becomes 0 and a learned mask vector is added; a token carrying a fused value is masked as a number or as a word with probability 0.5 each. The loss is the cross-entropy on the masked words plus λ times the mean squared error of a linear head that predicts the signed logarithm u of each masked value, clipped to ± 12. The word head is a $\mathrm{d} \times \mathrm{d}$ projection without bias, a GELU, a LayerNorm and a decoder tied to the token embedding W_{tok}, with a bias of its own. The specification set $\lambda = 0.5$; every training script passes $\lambda = 0.05$, because at 0.5 the number error, with $|u|$ up to 74 on web text, kept the token loss on its unigram plateau (`BENCH.md`, section "s2 training runs"). The s6 runs read 256-token sequences in batches of 64, and s3-pico 128-token sequences in batches of 128, 10% of them numeric rows. At the end, the state encoder starts from the pretrained stack and the static encoder from a copy of its first blocks; the fusion head and the scorer start untrained (`crates/sky-train/src/s2/train.rs`).

4.3 Decision training

In short. A loss at every exit, the last weighted twice the first, keeps every exit usable; passing loop dropout to one compiled graph as exit weights, with batch shapes rounded up to 32 tokens, made s3-pico's decision step 4 to 5 times faster.

Each exit e of the N_x exits gets a soft-target cross-entropy, $-\sum_i y_i \log p_i^{(e)}$ at temperature 1, with options that are only padding masked out, and the exits are weighted by

$$\mathcal{L} = \sum_e w_e \cdot \mathrm{CE}_e, \qquad w_e \propto 0.5 + 0.5 \cdot \frac{e-1}{N_x - 1}, \qquad \sum_e w_e = 1, \tag{6}$$

so the last exit weighs twice the first (`crates/sky-train/src/s2/config.rs`). There is no label smoothing. With probability 0.25 in every script (the *loop dropout*), a batch is supervised only on its first j exits, j drawn uniformly from 1 to $N_x - 1$, the encoder stops at that exit and the weights are renormalised over the exits kept. The draw enters the compiled graph as an input of exit weights, so one graph serves every outcome; together with batch shapes rounded up to multiples of 32 tokens, a change made at the same time, this made the s3-pico decision step 4 to 5 times faster than one graph per shape (`BENCH.md`, section "skycmd-s3"). The code also holds a self-distillation of the early exits from the last one and the softened targets of s4; neither is used by the files of this report.

Records are drawn with replacement, a source group by its weight and then an item uniformly, and questions that repeat within a batch share one static stream. The s6 runs also sort batches by state length in pools of 16, round shapes up to multiples of 32 tokens, and split a batch whose step would take more than 3 GB into parts whose gradients are averaged (`crates/sky-train/src/s2/data.rs`).

4.4 Optimizer and schedule

In short. AdamW on a warm-up, stable, decay schedule counted in working time trains every s6 file; Muon, tested in a pre-registered ablation, gained 0.0051 of validation accuracy where 0.01 was asked, and lost Flappy score.

Every s6 model is trained with AdamW [42] only: $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\varepsilon = 1\text{e-}8$, decoupled weight decay 0.01 on the matrices but not on the embeddings, norms and vectors, and the gradient clipped to a global norm of 1 (`crates/sky-train/src/optim.rs`). The schedule is warm-up, stable, decay: the rate rises linearly over the first 2% of the budget, stays flat until 80%, then falls linearly to 0.1 times its peak, where progress is working time over the budget.

The peak rate is the phase's default (3e-3 for pretraining, 1e-3 for decision training) times a recipe scale of 2.0, times $\sqrt{96/d}$ to move the rate found at width 96 to width d, times a per-size pilot scale chosen among 0.5, 1 and 2 (1 for s6-nano, 0.5 for s6-micro) (`scripts/train-s6.sh`, `docs/big-run-report.md`); Table 1 gives the rates this yields, computed here from that formula. The training seed is 0 unless Table 1 says otherwise.

The code has had two recipes of Muon [43] for the encoders' hidden matrices, and only s3-pico was trained with one of them. Appendix A.4 gives both. Before any s6 size was trained, a pre-registered ablation at the pico size, three seeds per arm, tested Muon (recipe 2) and seven other additions; each needed a validation gain of 0.01 (0.005 for the training options) without losing a game, and none was adopted: Muon gained 0.0051 and lost Flappy score (`runs/recipe/recipe.json`; companion report [1], section 6.7). So no s6 file uses Muon, and no published file uses recipe 2.

4.5 DAgger on the games

In short. Letting the model play and the oracle label the states it reaches took s3-pico from 145.3 Flappy pipes with 14 crashes in 30 episodes to 200 of 200 with none, after a first round that fell to 5.4 pipes; the number of rounds is fixed in advance.

A model trained on the oracle's play only sees the oracle's states; once it plays, its small mistakes take it where the oracle never goes. DAgger [12] lets the model play and asks the oracle to label the states it reaches. Algorithm 1 is what `scripts/train-s6.sh` and `scripts/specialize-s3.sh` run, with the collection in `crates/sky-arena/src/main.rs` and `crates/sky-games/src/gen.rs`.

Algorithm 1 DAgger rounds as implemented.

Input: the checkpoint θ_0 of the decision training and its ternary file M_0, calibrated by `s2-calibrate`; R rounds (3); N records per game and round (15,000); the oracle share μ_1 (0.2); the round's budget in working minutes and its mixture (Table 1 and below).

1: **for** each round $r = 1, \ldots, R$ **do**

2: For each game, on arena seed r: play episodes with M_{r-1} in the native engine, at the exit thresholds stored in the file, Flappy episodes stopping at 12 pipes. At every decision, record the state with the oracle's target distribution; then the oracle acts with probability μ_r (its best option), and otherwise the model acts (its most probable option). Keep records through the class-balancing filter until N are kept. This gives D_r.

3: $\theta_r \leftarrow \theta_{r-1}$ trained for the round's budget on the base mixture plus D_r, ternary on every step, with a new optimizer state and schedule.

4: Fit the temperatures and the thresholds of section 5.2 on validation, export M_r and play 30 episodes per game at full depth, for the record only.

5: s6 models: $\mu_{r+1} \leftarrow \mu_r/2$. s3-pico: $\mu_{r+1} \leftarrow \mu_r$.

6: **end for**

7: **Output:** θ_R and M_R, the last round, the checkpoint rule fixed in advance.

Four choices depart from the textbook. The oracle keeps a share μ of the decisions (β in the DAgger paper), the mixed policy of the original paper applied per decision, so episodes still reach the later parts of each course. Each round trains on its own records and the base mixture, not on the union of all rounds; an aggregated variant was tried in the night experiments and not kept (`docs/night-experiments.md`). The closest named method to this per-round scheme is SMILe [57], which also trains on each iteration's data only and lets the expert act with a share that decays geometrically; unlike SMILe, Algorithm 1 fine-tunes one network rather than mixing the policies of every iteration. The mixture keeps earlier data: games 0.5, the round 0.35, numeric 0.05 and decisions 0.1 for s3-pico; games 0.4, the round 0.25, decisions 0.1, teacher labels 0.2 and numeric 0.05 for s6-nano and s6-micro. And the episodes are played at the thresholds the file carries, which are those fitted on held-out questions by the rule of section 5.2, not those of the published file: s6-nano's last round held 0.5 for each type before its thresholds were set to 1 for publication (`BENCH.md`, section "skycmd-s6"), so its rounds collected states that early exits reach while the file ships at full depth.

On s3-pico, round 1 collapsed: Flappy fell from 145.3 pipes with 14 crashes to 5.4 pipes with 30 crashes in 30 episodes, while the Dodge vetoes fell from 9,137 to 4,205; rounds 2 and 3 reached 200 of 200 pipes with no crash, and the vetoes 1,030 and then 14 (`BENCH.md`, section "DAgger rounds"). Each round had 8 working minutes.

4.6 Budgets and the brake

In short. Budgets count working minutes, so the brake's rests do not change what a run computes; other jobs on the machine still change how many steps a budget buys (section 9).

Every run shares the laptop with its owner. A *brake* lets it run hard when the machine is idle and gently when it is in use, hot or on battery (`docs/idle-training.md`, `crates/sky-train/src/brake.rs`). A daemon writes a duty cycle η and a work and rest cycle to a file that the trainer re-reads every 5 seconds: 0.5 with 45 minutes of work and 15 of rest when the machine is in use, hot or on battery, 0.85 with 50 and 10 when it is left alone, and 0.95 with 55 and 5 at night. Before each step the loop sleeps $(1/\eta - 1)$ times the time worked since its last rest, and it naps through the rest part of the cycle. Every rest is added to the start of the run's clock, so the clock counts working time only: a budget of 90 minutes computes the same steps in every mode, and only the wall-clock time changes. The last checkpoint is the result, except where a pre-registration named the moving average of the weights (section 6); no checkpoint is chosen by its validation score.

4.7 Recipes

In short. Table 1 gives every setting of seven published files, so that each can be trained again from its column.

Table 1 gives the settings of seven published files.

Two cells need a word. s6-micro (v2) kept v1's pretraining and doubled the decision training (`BENCH.md`, section "skycmd-s6"); its QAT cell comes from its script, which passes q = 0.9 (section 3.6). No file records the total of s3-pico's training data; tutorial, chapter 04 counts 210,000 game records for the s1 and s2 models and 100,000 Doom records added for s3-pico.

5 Calibration and early exits

5.1 Temperatures

In short. A temperature per exit, type and bucket of option counts, fitted on validation, calibrates every exit, so that one threshold can be compared with the probabilities of any exit.

After training, a temperature is fitted for each (exit, type) pair and, where a bucket holds at least 50 validation questions, for each (exit, type, bucket), on the validation split, for the float and the ternary weights separately (`crates/sky-train/src/s2/eval.rs`). The bucket is the number of options: 2 or fewer, 3 to 5, 6 to 10, 11 and more. The fit minimises the negative log-likelihood against the soft targets by a golden-section search on ln T over [0.05, 20], 80 iterations (`crates/sky-train/src/metrics.rs`). The values go into the file under `temperature.e{exit}.{type}[.{bucket}]`. The expected calibration error (ECE) is measured on the top probability with 15 equal-width bins, as in Guo et al. [40], per source and over all sources pooled.

5.2 Thresholds fitted on held-out questions

In short. The first rule fits thresholds on held-out questions, and it failed where it matters: at its thresholds s2-nano crashed in every Flappy episode on five seeds.

The first rule, in the specification of s2, fits one threshold per type on validation questions: the smallest τ in 0.50, 0.51, …, 0.99 whose early-exit accuracy is within 0.005 of full depth, and 1 if none is (`crates/sky-train/src/s2/eval.rs`). For s2-nano it gave 0.62, 0.62 and 0.69 for choice, score and noul. Played at those thresholds, over 5 arena seeds of 30 episodes, s2-nano crashed in every Flappy episode (15.86 ± 2.30 pipes of 200); the same weights at τ = 0.95 flew 191.94 ± 7.16 pipes (`loopcal.json`; companion report [1], section 5.2). The held-out questions come from the oracle's play, and the states a model reaches when it plays with early exits are not those. This rule still writes the thresholds of every file that `s2-calibrate` produces, the DAgger rounds' included (section 4.5).

5.3 Thresholds chosen by playing

In short. Thresholds chosen by playing the games keep their scores: s3-pico keeps 200 of 200 Flappy pipes at 1.44 of its 4 passes, and even at τ = 0.5 it answers Flappy in 1.0 pass and still flies 200 of 200, over five seeds (`loopcal.json`).

Table 1. Recipes of seven published files. Parameters and bytes from the manifest; shapes from the configuration files and `BENCH.md`; budgets and steps from `BENCH.md` (section of each model) unless noted. Block passes are blocks × loops; windows are state / head tokens. Times are working minutes. Batches, optimizers, rates, seeds and mixtures are read from the training scripts and the run arguments (`runs/s6-ear2-nano/args.json`, `runs/s6-cast2-pico/args.json`); the peak rates of s6-nano, s6-micro, s6-route-nano and s6-secrets-verify-pico are computed here from the formula of section 4.4, and s6-ear2-nano's is its pilot's choice (`runs/s6-ear2-pilot/choice.json`). QAT gives the part of the decision training run ternary (section 3.6): the last 90% is a float warm-up of 10%.

	s3-pico	s6-nano	s6-micro (v2)	s6-ear2-nano	s6-route-nano	s6-secrets-verify-pico	s6-cast2-pico
parameters	445,345	1,486,225	5,587,201	1,306,178	5,587,201	445,345	377,568
file bytes	251,104	715,136	2,420,032	631,936	2,418,912	249,984	148,512
width d / ff	96 / 192	144 / 288	256 / 768	160 / 640	256 / 768	96 / 192	96 / 192
block passes	1 × 4	2 × 2	2 × 3	4 × 1	2 × 3	1 × 4	2 × 2
vocabulary	1,024	2,048	4,096	none: 43 audio channels per 10 ms	4,096	1,024	none: patches of 32 points
windows	64 / 96	512 / 160	384 / 128	4 s, 50 positions of 80 ms	384 / 256	320 / 192	2,048 points in, 64 out per pass
attention heads	1	2	4	2	4	1	2
training data	not recorded as one total: the game records of chapter 04, decision sets, numeric curriculum, web text	0.87B pretraining tokens; 2,546,304 decision draws (study)	0.52B pretraining tokens (v1's); 65,471 decision steps of 64	270,946 clips of `smart-turn-data-v3.2-train` (pre-registration)	81,596 tool, 95,411 intent and 14,621 When2Call [58] records	CredData 49,864, issue reports 40,057 and synthetic 40,000 items read	synthetic series, real series and 1,011 USCRN series (8,282,112 points)
pretraining	8 min, 4,814 steps	90 min, 53,360 steps	180 min, 31,700 steps (v1's)	none	from s6-micro's	12 min, 11,296 steps, from v1's stack	none
decision training	15 min, 6,440 steps	90 min, 19,893 steps	360 min, 65,471 steps	240 min, 305,700 steps	120 min, 20,504 steps	30 min, 14,117 steps	720 min, 377,082 steps
DAgger rounds	3 × 8 min: 4,683, 5,136, 5,313 steps	3 × 15 min: 4,645, 4,660, 4,644 steps	3 × 20 min: 3,822, 4,673, 4,969 steps	none	none	none	none
batch: pretraining / decision	128 sequences of 128 tokens / 128	64 of 256 / 128	64 of 256 / 64	none / 128 clips	s6-micro's / 64	64 of 256 / 64	none / 256 windows
optimizer	recipe 1: Muon on the hidden matrices, AdamW elsewhere	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
peak rate: pretraining / decision	AdamW 3e-3 and Muon 0.02 / AdamW 1e-3 and Muon 0.01	4.90e-3 / 1.63e-3	1.84e-3 / 6.12e-4	none / 7.745e-4	s6-micro's / 6.12e-4	6e-3 / 2e-3	none / 3e-3
seed	0; DAgger arena seeds 1, 2, 3	0; DAgger arena seeds 1, 2, 3	0; DAgger arena seeds 1, 2, 3	1	0	0	1
first decision mixture	games 0.6, numeric 0.1, decisions 0.3	decisions 0.25, teacher labels 0.30, games 0.35, numeric 0.10	as s6-nano	one source	tools 0.60, intents 0.25, When2Call 0.15	CredData 0.45, issue reports 0.35, synthetic 0.20	real series in 60% of the windows
QAT	last 90%; DAgger 100%	last 90%; DAgger 100%	last 90%; DAgger 100%	last 15%	last 90%	last 90%	last 20%
teacher	none (oracles)	GLM-5.3-flash, 747,077 labelled, 560,238 kept	GLM-5.3-flash, the same labels	Smart Turn v3.2	GLM-5.3-flash wrote descriptions (4,174 calls), no labels	none	TinyCast (our port)

The closed-loop rule (`scripts/calibrate-exit.sh`, Algorithm 2) plays the games at full depth, then at every candidate threshold on the same episodes, and keeps the smallest candidate whose games stay within 2% of full depth with no extra crash or veto. A *veto* is the coded safety reflex of the Dodge game taking over from the model (tutorial, chapter 20).

Algorithm 2 Closed-loop exit calibration as implemented.

Input: a ternary file M; the games G (Flappy, Rescue and Dodge); n_{ep} episodes per game (20) on seed 7; the candidates 0.5, 0.6, 0.7, 0.8, 0.85, 0.9, 0.95 and 0.98.

1: Play n_{ep} episodes of each game at $\tau = 1$. For each game g, keep the reference $A_g(1)$: mean score, rescue rate, completion rate, crashes and vetoes.
2: For each of the eight candidates τ: play the same episodes with τ as the threshold of every question type, giving $A_g(\tau)$.
3: Accept τ if, for every game g and each metric c among the mean score, the rescue rate and the completion rate, $A_{g,c}(\tau) \geq A_{g,c}(1) - 0.02\,|A_{g,c}(1)|$, and $A_g(\tau)$ has no more crashes and no more vetoes than $A_g(1)$.
4: **Output:** the smallest accepted τ, or 1 if none is, written as the threshold of the three types.

Every candidate is played, and the rule keeps the smallest one accepted. Nothing makes acceptance monotone in τ: a candidate can be accepted while a larger one is rejected, and the output is then the smaller.

For s3-pico the rule accepted $\tau = 0.6$ and rejected 0.5 for one extra veto, 5 against 4 in the 20 Dodge episodes; at 0.6 the model used 1.44, 1.48 and 1.05 of its 4 passes in Flappy, Rescue and Dodge over those 20 episodes (`BENCH.md`, section "Closed-loop exit calibration"). The rule chooses and checks on the same seed. A separate evaluation, `cargo xtask loopcal`, replays fixed thresholds on 5 seeds (7, 11, 13, 17 and 19) of 30 episodes and chooses nothing; on s3-pico's four games it found the same scores and no crash at $\tau = 1$, 0.6 and 0.5 (Dodge vetoes 9.6 ± 4.1 at 1 and 0.6, 9.8 ± 4.4 at 0.5), and it produced the s2-nano numbers of section 5.2 (`crates/xtask/src/loopcal.rs`, tutorial, chapter 21). A one-veto difference is thus well within the seed-to-seed spread of s3-pico's Dodge vetoes, ± 4.1 over five seeds: the rejection of 0.5 does not show that 0.5 plays worse.

5.4 A conformal variant, checked by playing

In short. Conformal thresholds bound how often an early answer differs from the full-depth one per decision, not per episode, and the check by playing rejected them for s6-nano and for s6-micro v2.

For s6-nano and s6-micro, the thresholds came from conformal risk control [30], in the early-exit setting of CALM [31] and Fast yet Safe [32]; the code credits SAFE-KD [33], which calibrates per-exit stopping thresholds the same way (`crates/sky-arena/src/crc.rs`). The games are played at full depth, and every decision records, for each exit, its top calibrated probability and its answer. For a candidate τ, the loss of a decision is 1 when the answer at the exit that τ selects differs from the full-depth answer, and $\hat{R}(\tau)$ is its mean over the n_{dec} decisions of a type. For each type, the scan goes down from 1.0 to 0.50 in steps of 0.01 and keeps the smallest τ such that

$$\frac{n_{\text{dec}} \cdot \hat{R}(\tau') + 1}{n_{\text{dec}} + 1} \leq \alpha \quad \text{for } \tau \text{ and every larger } \tau' \text{ of the grid,} \tag{7}$$

with $\alpha = 0.02$ in the s6 script. A loss that counts disagreements with the full model is that of CATs [29], which calibrate their stopping rule to keep early answers consistent with the full model. The bound holds when new decisions are exchangeable with the calibration decisions. Two things stand against that here: the calibration decisions were made at full depth, and the decisions of one episode follow one another along a trajectory, so they are not exchangeable even at full depth; n_{dec} counts decisions, not the 30 episodes per game that produced them.

The pre-registered rule then plays the games at the chosen thresholds, on the same seed and episodes as the reference, with the acceptance test of Algorithm 2, and drops the thresholds if they fail (`scripts/train-s6.sh`). For s6-nano it rejected them: the model used 1.01 to 1.83 of its 4 passes per game, but Rescue fell to 2.80 of 3 and the Dodge vetoes rose from 23 to 1,016. It accepted the thresholds of s6-micro-v1, at which every game kept its mean score; that file then failed its gate on calibration at both depths. It rejected those of s6-micro v2 by one veto, 21 against 20, with the same scores, within the seed-to-seed spread of section 5.3 (`BENCH.md`, section "skycmd-s6", Arena; companion report [1], section 5.2). s6-nano and s6-micro ship at full depth, with a threshold of 1 for each type. Why the fitted thresholds failed for s6-nano is a hypothesis, not a measurement: playing with early exits changes the states visited, so a guarantee fitted on full-depth play need not cover them, as section 5.2 found for thresholds fitted on held-out questions.

6 Task-specific systems

6.1 End of turn

In short. A 631,936-byte student of Smart Turn v3.2 that reads 4 s instead of 8; its shape is the first of four sizes whose WebAssembly median was at most 4.0 ms, and its pilot kept distillation over hard labels (validation 0.8715 against 0.8437).

The end-of-turn detector answers one question: has the speaker finished? Its graph, `skycmd-turn`, reads the last 4 seconds of 16 kHz audio (`crates/sky-infer/src/audio.rs`, `crates/sky-infer/src/turn/`, `docs/s6-ear2-prereg.md`).

- **Front end.** Each 10 ms hop of 160 samples reads a 40 ms span; a periodic Hann window of 25 ms (400 samples) centred in it gives a 512-point FFT, and 40 triangular mel filters from 20 to 8,000 Hz on the power spectrum give log10 energies, floored at 1e-8. Three prosody channels follow. The fundamental frequency f0 and the voicing come from the autocorrelation of the 40 ms span, computed as the FFT of the power spectrum and normalised by the window's own autocorrelation, searched over lags of 40 to 200 samples, that is 80 to 400 Hz, with a small octave cost and a parabolic refinement, after Boersma's lag-domain method [59]; the voicing is the peak, clamped to [0, 1], and the f0 channel is log2(f0 / 100) when the voicing reaches 0.45, else 0. The energy channel is the log10 of the summed mel energies. Each hop thus has 43 channels.
- **Normalisation.** Each 4 s window is normalised as Whisper's feature extractor does [60]: with M the largest log-mel value of the window, each mel value x becomes (max(x, M − 8) − M + 4) / 4, and the energy (x − M + 4) / 4; f0 and voicing stay as they are. The window keeps the 400 most recent hops.
- **Network.** Eight hops (80 ms) form one position of 344 values, so 4 s are 50 positions. A float projection with bias maps each position to width 160, then 4 blocks (pre-norm, fused projections without bias, rotary embeddings, full attention and a GELU feed-forward of width 640, 2 heads) with ternary matrices, then an attention pooling over the positions p, $\psi_p = W_2 \tanh(W_1\ h_p + b_1) + b_2$, a = softmax($\psi$), $\bar{h} = \sum_p a_p\ h_p$, and a two-layer head with one logit z. The output is $\sigma(z / T_{\text{turn}})$, with $T_{\text{turn}} = 1.4947854$ stored in the file. The shape is the fourth of four sizes, chosen before training as the first whose WebAssembly median was at most 4.0 ms (`BENCH.md`, section "s6-ear2-nano").
- **Distillation.** The teacher is Smart Turn v3.2 [61], which reads 8 s; the student reads 4 s. The loss is $0.5 \cdot \text{BCE}(y, \sigma(z)) + 0.5 \cdot \text{BCE}(\sigma(z_{\text{tch}}), \sigma(z)) + 0.25 \cdot (1 - \cos(P\,\bar{h}, \text{LN}\,\bar{h}_{\text{tch}})) + 0.5 \cdot$ (the mean over 25 steps of $1 - \cos(P'\,\bar{h}_{\text{step}}, \text{LN}\,H_{\text{tch,step}})$): the hard label y, the teacher's probability, the teacher's 384-dimensional pooled embedding $\bar{h}_{\text{tch}}$, and its encoder states pooled to 25 steps H_{tch}, with the student's positions averaged into the same 25 steps; the projections P and P′ exist only in training (`crates/sky-train/src/turn/model.rs`). Training also shortens the context at random and masks a few mel bands and hops. The pilot kept distillation over hard labels (validation 0.8715 against 0.8437), and the run exported the moving average of the weights at a decay of 0.999.
- **Two thresholds.** The file stores $\delta_{0.5} = 0.5$, the threshold the accuracy is read at, and $\delta_{85} = 0.724398136$, the highest validation threshold whose recall of finished turns reaches 0.85; the detector answers "finished" when $\sigma(z / T_{\text{turn}}) \geq \delta$.

Gate, reading and limit. Accuracy of at least 0.85, and at most 0.08 under its teacher's, on the 31,527 test clips of Smart Turn v3.2: s6-ear2-nano reads 0.8993, in 2.39 ms per decision in WebAssembly, and passed; on eot-bench, real speech addressed to agents, its AUC is 0.7299 against its teacher's 0.7838, and at δ_{85} it answers "finished" on 0.5253 of the real ends of turn, so it needs a fallback timer (companion report [1], section 6.4). Code and policy: Use it for real.

6.2 Tool routing

In short. A trained model that reads the options and each tool's BM25 score, multiplied option by option with a static embedding: alone, the model part reads 0.7926 and the embedding 0.7386 on 945 fresh requests; multiplied, 0.8868 (`runs/s6-nano3/eval-fresh/results.md`).

The router picks one tool from a list given at run time, or *none* (`crates/sky-infer/src/route.rs`, `crates/sky-infer/src/embed.rs`, `crates/sky-web/src/router.rs`).

- **The model part (s6-route-nano).** One `choice` question, "Which tool should handle the user's request?", whose

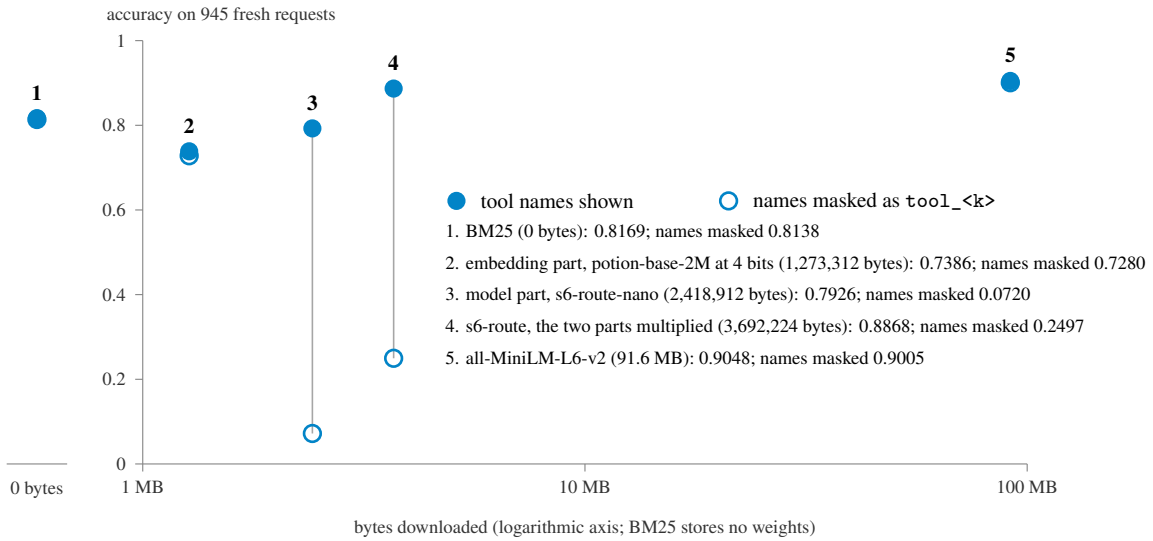


Figure 2. The router size map: accuracy on the 945 fresh requests of s6-route's gate against the bytes each router downloads, with the tool names shown (filled) and replaced by numbered placeholders (hollow). Alone, the model part reads 0.7926 and the embedding part 0.7386; multiplied, 0.8868, under the 0.9048 of all-MiniLM-L6-v2, which takes about 25 times the bytes (computed here), and only while the names are shown. BM25 stores no weights and is drawn at its own 0-byte position. The numbers come from `runs/s6-nano3/eval-fresh/results.md`; the figure is written by `scripts/paper-methods-figure.py`, and its text alternative is in the SVG.

options are the tools as `name: description` and, always last, `none of these tools fits the request`. The state is `User: <request>`, then one line `BM25: <name> score <s> rank <r>; …` with each tool's BM25 score against the request, written with two decimals, and its rank in the slate. BM25 is the Lucene variant of BM25S 0.3.16 [62] with k1 = 1.5 and b = 0.75, over the slate's own tools as documents, on lower-cased runs of two or more word characters without 33 English stopwords. The scores go into the state, not the options, because the options are encoded once per tool list and cached (section 3.3); each entry finds its option by the name, and its two numbers fuse onto `score` and `rank` (section 3.2). The model has the s6-micro graph with windows of 384 state and 256 head tokens, and its training data hold counterfactual slates, with the right tool removed or every tool renamed, so that only reading the options lowers the loss (`BENCH.md`, section "s6-route-nano").

- **The embedding part.** potion-base-2M [63], a Model2Vec [64] static embedding of dimension 64 and 29,528 tokens, is packed to 4 bits: two codes per byte in −7 to 7 and one 16-bit scale per token, max |v| / 7, in a 1,273,312-byte file with its tokenizer. A text's WordPiece tokens are averaged and the mean is L2-normalised. The request's cosine with each `name: description` is computed, *none* gets the constant $c_{\text{none}} = 0.39611971$, and $p_{\text{embedding}} = \text{softmax}([\cos_1, \ldots, \cos_n, c_{\text{none}}] / T_{\text{emb}})$, with $T_{\text{emb}} = 0.0671021167$. Both were fitted on the validation slates when this part was a baseline of s6-route-nano: c_{none} maximises the validation accuracy, the lowest value among ties, and T_{emb} minimises the validation negative log-likelihood, by a grid on ln T_{emb} and then a golden-section search (`scripts/route-baselines.py`).
- **Fusion (s6-route).** No network is trained for the fusion; four scalars of the router were fitted on validation slates, c_{none} and T_{emb} above, and κ and δ_{ask} here. The two distributions are multiplied option by option and renormalised, $p(i) \propto (p_{\text{model}}(i) \cdot p_{\text{embedding}}(i))^{\kappa}$, with $\kappa = 0.81$, the minimum of the validation negative log-likelihood on a grid from 0.01 to 4.00 (`scripts/s6-route3-fit.py`). The router asks the user instead of answering when the fused top probability is below $\delta_{\text{ask}} = 0.6710473485$ (`tau_ask3` in the code), the smallest threshold at which accuracy among the answered validation slates reaches 0.95. The model part runs at full depth, or stops at its first exit in a fast mode, and the engine caches the static stream and the tool embeddings of the last tool list.

Gate, reading and limit. s6-route-nano had to beat BM25 on unseen tools and tied it (0.8395 against 0.8500), and failed; s6-route, a second look read on 945 fresh requests, reads 0.8868 against 0.8169 for BM25 and 0.9048 for all-MiniLM-L6-v2, and passed, but falls to 0.2497 when the tool names are masked (companion report [1], section 6.3). Code and policy, and when BM25 is the better choice: Use it for real.

6.3 Secrets: a verifier of rule candidates

In short. Rules propose, a 249,984-byte model judges each candidate in context, and the filters a rule scanner usually applies become the model's features.

The verifier judges one candidate at a time: a rule proposes a value, and the model says whether it is a real credential (`crates/sky-infer/src/secret_cand.rs`, `crates/sky-web/src/verify.rs`, `docs/s6-secrets2-prereg.md`). This is the design of CredSweeper [14], Credential Digger [65] and FP Remover [15].

- **Candidates.** The rules are those of secretgate [66], a scanner by the author of this report, in a candidate mode: the 222 rules it takes from Gitleaks [67] and its own, with the entropy gates, the placeholder test, the stopwords and the allowlists turned off, since those are the filters the verifier is meant to replace, plus a rule for a credential keyword in a name followed by a value of 4 to 256 characters and, after the one revision allowed, a rule for authorisation schemes (`BENCH.md`, section "s6-secrets-verify-pico"). Each filter that was turned off becomes a 0 or 1 feature of the candidate.
- **The state.** One part per line, in this order, since a state is truncated from its end: `rule` and its id; `path` with at most 4 scope words taken from the file path (the 57 words of CredData's `word_in_path.json`, such as test or example) and the extension, not the path itself; `key` and the variable name, at most 40 characters; `value` and the value, whole up to 64 characters, else its first and last 32; the features `len`, `ent` (entropy in hundredths of a bit), `cpt` (characters per token, in hundredths), `dig`, `up`, `low` and `oth` (character counts), `word` (the longest run of letters that is a dictionary word, the dictionary being a word list of 1,706,770 bytes published with the model, manifest) and the four filter flags; `line`, the value's line with the value replaced by `<V>`, at most 120 characters centred on it; and `prev` and `next`, one line before and one after, at most 100 characters each. Each feature is a `name value` pair, so with value fusion it costs one token. The state window is 320 tokens.
- **The question and the thresholds.** The gated question is a `noul`, "Is this value a real credential that should be revoked?", and P(real) is its probability of `true` at full depth. A second question, reported and not gated, is a `choice`, "What kind of value is this?", over `a real credential`, `an example, test value or placeholder` and `not a credential`, CredData's three labels. The model is the s3-pico graph. The file stores three thresholds, each maximising an F-score on the full-depth P(real) of the CredData validation candidates before the test was opened: $\delta_{F1} = 0.2598114610$ for F1, $\delta_{F0.5} = 0.4684373736$ for F0.5 and $\delta_{F2} = 0.0780794695$ for F2 (the keys `tau_f1`, `tau_f05` and `tau_f2`). A candidate is accepted when P(real) $\geq \delta$.

Gate, reading and limit. A line F1 of at least the best rule scanner's plus 0.15 (0.8832) on 221 test repositories: s6-secrets-verify-pico reads 0.8561, above every rule scanner (0.7332 for the best), and failed its margin (companion report [1], section 6.2). Code: Use it for real, where secretgate runs it as an opt-in experiment.

6.4 Forecasting

In short. The same blocks on patches of a series, with the period detector of TinyCast as an input and an early exit whose tolerance is set by a conformal bound.

The forecaster, `skycmd-cast`, reuses the state blocks on patches of a time series (`crates/sky-infer/src/cast/`, `crates/sky-train/src/cast/`, `docs/SPEC-S6-CAST.md`).

- **Input.** The last 2,048 points are scaled to [0, 1] by their minimum and maximum, and cut into 64 patches of 32 points aligned on the last point; a missing value is 0 with a mask bit. Each patch row holds its 32 values and 32 mask bits, and is projected to width 96.
- **Encoder and decoder.** 2 blocks applied twice, with an exit after each block pass. Two learned queries, each standing for 32 future steps, attend to the encoder's output through 2 cross-attention blocks, as in the fusion head; a head gives 32×9 outputs per query, made monotone in the quantile level by adding cumulative softplus increments above and below the median. The nine levels are the deciles 0.1 to 0.9.
- **Loss.** The pinball loss, $\max(\nu(y - \hat{y}), (\nu - 1)(y - \hat{y}))$ for level ν, target y and predicted quantile $\hat{y}$, averaged over observed steps and levels, at every exit with the weights of equation (6); plus the mean absolute gap to the quantiles of TinyCast [68] on 50% of the windows, with a weight annealed from 0.5 to 0.
- **The periodicity input (s6-cast2-pico).** TinyCast's period detector [69], which has no parameters, finds up to 4 significant periods in the context from its periodogram. Each patch gets the sine and cosine of the phase of each

period at its last point, 8 channels, and each query gets a seasonal draft, the 32-step lag copy and the 32-step phase-binned mean of the first period, projected and added to it.

- **Early exit.** A pass stops when the median of an exit differs from the previous exit's by less than a tolerance at every step; the tolerance, 0.012839192 for s6-cast2-pico, is the largest whose early median stays within 0.02 of the full-depth median under a conformal bound at $\alpha = 0.05$, on 2,000 synthetic windows.

Gate, reading and limit. GIFT-Eval [70] nGMASE of at most 0.838 and nWQL of at most 0.587: s6-cast2-pico reads 0.8847 and 0.6190, and failed, behind the project's port of TinyCast; GIFT-Eval is partly in-sample for its design (companion report [1], section 6.5). No forecaster is in Use it for real.

7 The engine

Two graphs in one language. Training runs in Rust through mlx-rs 0.32.0 on the laptop's GPU; inference runs in plain Rust on the CPU, natively and compiled to WebAssembly (`crates/sky-infer/`). The graph is thus written twice, and golden files tie the two: `sky-train s2-golden` writes a small random model with the probabilities of every exit, and the engine must match them within 1e-4 for float files and 1e-3 for ternary ones; on the s2, s3 and temporal-carry goldens it matched within 6.3e-6 (`BENCH.md`, section "Inference"). The golden tests also check that the engine's tokenizer gives the token streams of Hugging Face `tokenizers`.

GGUF as a container. Every file is GGUF version 3, with metadata keys and tensors (`crates/sky-gguf/`). Only the s1 files, which use llama.cpp's `modern-bert` architecture with the `laya` decision head, also run in llama.cpp and wllama; there, their probabilities agreed with the engine's within 4.7e-4 and 1.1e-4 (`docs/compat.md`). Every other architecture is read only by this engine: `skycmd-s2` for s2 to s6, `skycmd-iso` for s5, `skycmd-turn`, `skycmd-cast` and `skycmd-embed`.

Kernels. On these small matrices, float kernels beat packed ternary ones: on a 288×96 matrix with 9 rows, the shape of s3-pico's attention projection on a 9-token state, the float panels take 3.99 µs per product natively and the packed add and subtract kernels 12.23 µs, so the 2 bits are a storage format. At load, the 2-bit matrices are expanded to 32-bit float: the 2-bit form is what is downloaded and stored, not what is multiplied (`crates/sky-infer/src/s2/engine.rs`). The float matrices are repacked in panels of 8 outputs for an outer-product kernel that keeps 16 to 24 accumulators whatever the number of rows; the first block reads a table of the token embeddings already multiplied by its projection, with a 17-input correction for value tokens; and the fusion keys and values of the state are folded into the option queries (`BENCH.md`, section "Inference"). Three opt-in kernels compute on the ternary codes instead (`crates/sky-infer/src/s2/engine.rs`, `TernaryCompute`). Packed add and subtract kernels keep the 2-bit weights in memory, the smallest footprint, and are slower than the float panels, as measured above. Lookup tables of the 16 subset sums of each group of 4 inputs, in the style of T-MAC [71] and bitnet.cpp [72], beat the packed kernels but not the float panels on these small matrices, which stay in the first-level cache, and need two activation passes to stay within 3.5e-4 of the goldens. Int8 dot products are the fastest path natively, 21 against 25 µs per Flappy decision for s3-pico, at up to 5.5e-4 from the goldens. `TernaryCompute::Auto` expands to float everywhere, so the float path is the default; the browser module exposes only the packed kernels, as an option of `withOptions` (`crates/sky-web/src/lib.rs`), and has no option for the int8 or lookup paths (`BENCH.md`, section "Inference").

What the cache saves. For s6-route, under the load given in Table 2, the first request on a tool list, which encodes the list, took 45,279 µs in WebAssembly, and the later requests on the cached list 30,879 µs at the median (`BENCH.md`, section "s6-route (s6-nano v3)").

Two WebAssembly builds. `scripts/build-wasm.sh` compiles the module twice, with SIMD128 and with SIMD128 plus relaxed SIMD, whose fused multiply-add the float kernel uses. The page loads the relaxed build when `WebAssembly.validate` accepts a relaxed instruction, and the SIMD128 build otherwise (`site/.vitepress/theme/lib/wasm.ts`).

Measurements. Table 2 gathers the agreement between the browser module and the native engine, and the latencies, as each result section measured them: on the Apple M5, under Node for WebAssembly, in sessions and under loads that differ from row to row, as the sources state.

Table 2. Agreement between WebAssembly and native probabilities, and latencies, in µs (medians). WebAssembly is the SIMD128 build under Node unless a second value is given for the relaxed build. Sources: `BENCH.md`, the section of each model. Each row is one measuring session with its own inputs and load, so latencies compare within a row, not across rows: s6-route's 30,879 µs, below the 38,929 µs of s6-route-nano, its own model part, comes from another session on other requests.

model	what is timed	session and load	native	WebAssembly	largest probability gap, same answers
s3-pico	one Flappy question, first exit (section "Inference")	best of 3 rounds; machine shared with training jobs	25	44; relaxed 37	
s6-nano	a long bench question, full depth	the first 200 bench questions; load not stated	7,385	14,600 to 15,200	9.5e-7, 200 of 200 bench questions
s6-nano	a Flappy frame, two questions and physics	`bench-node.mjs --game flappy`; load not stated		436	
s6-secrets-pico	one bench decision, full depth	14/11/2026, a GPU training running	789 and 786	1,420 and 1,427	1.2e-6, 200 of 200
s6-secrets-verify-pico	one candidate, full depth	GPU idle, before the baselines ran	1,606	2,666	4.17e-7, 200 of 200
s6-ear2-nano	one decision on a 4 s window, one thread	1,000 decisions, no training running	1,400	2,390	1.2e-7, 20 of 20 clips
s6-route-nano	one request, tool list cached	200 T-tools requests (`runs/s6-nano2/eval-test/latency/`)	24,031	38,929; relaxed 23,532	2.2e-6, 200 of 200
s6-route	one request, tool list cached	200 fresh T2-tools slates (`runs/s6-nano3/eval-fresh/latency/`); the owner at work and a GPU run of another model active	24,648 (model part)	30,879; relaxed 23,536	1.6e-6 against Python, 200 of 200
s6-cast2-pico	one 64-step forecast, one core	1,000 forecasts, after the training had ended	838	1,636	

8 Lessons for practitioners

Table 3 gathers what each choice bought where it was measured, how strong that evidence is, and the choices that were never measured. Most rows rest on one model and one training run, and the readings were made under the protocol of the companion report [1].

9 Limitations

- **One run per recipe.** Every file of Table 1 was trained once, on one seed. The intervals of the result sections are over test items, not over training runs, so nothing here says how much a recipe moves when trained again.
- **One machine.** Budgets count working minutes on one laptop, often shared with other jobs, so the steps a budget buys depend on the load, as s6-micro v2's DAgger rounds show (`BENCH.md`, section "skycmd-s6").
- **Little was ablated.** The pico ablation tested eight additions at one size and three seeds. Nothing separates the causes of s6-nano's gain over s4-micro (pretraining, teacher, tokenizer and windows changed together), the looped block against a stack of distinct blocks at equal size, or loop dropout, the numeric curriculum and the value fusion from their absence (tutorial, chapter 27).
- **Shortcuts in the data.** Methods that fit the training data can fit its accidents. In the router's training data, the right tool of every renamed slate was the only option whose description the teacher had rewritten, and the model learned to pick that description (`docs/s6-route4-diagnosis.md`); s6-nano answers each question of the bench almost as its most common training answer would (companion report [1], section 5.1).
- **Calibration chosen and checked on the same episodes.** The closed-loop rule and the conformal check choose and check on seed 7; the five-seed evaluation came after, for some models only. The conformal bound was fitted on full-depth play; the decisions of an episode are not exchangeable even at full depth, and still less once exits fire early; and n_{dec} counts decisions, not episodes. The one-veto differences that decided s3-pico's 0.5 and s6-micro v2's thresholds are within the seed-to-seed spread of the Dodge vetoes.
- **DAgger rounds played at thresholds the published file does not use.** The rounds play each file at the thresholds

Table 3. What each choice bought. The evidence column names the models, seeds and splits; a section number points to the measurement in this report.

choice	measured effect	evidence
Train the ternary weights in; do not round a float model	rounding cost 6 to 11 points of validation accuracy (pico 0.470 to 0.406, nano 0.542 to 0.432); 30% of the budget in QAT gave 0.482 and 0.556	the s2 models, two sizes, one run each (section 3.6)
Treat the 2 bits as storage, and benchmark float kernels first	3.99 µs per product for the float panels against 12.23 µs for the packed ternary kernels	one 288 × 96 shape, one CPU, natively (section 7)
If the model's answer changes its next input, choose the exits by running the loop	at thresholds fitted on held-out questions s2-nano crashed in every Flappy episode (15.86 pipes of 200); chosen by playing, s3-pico kept 200 of 200 at 1.44 of 4 passes	s2-nano and s3-pico on five arena seeds; s6-nano's conformal thresholds (Dodge vetoes 23 to 1,016) on one seed; four friendly games (section 5)
Do not read a per-decision α as an episode guarantee	at $\alpha = 0.02$, an episode of 21,300 Flappy decisions allows on average 426 early answers that differ from full depth	arithmetic, computed in the companion report [1], section 5.2; the decisions of an episode are not exchangeable (section 5.4)
Fix the number of DAgger rounds in advance	round 1 fell to 5.4 pipes with 30 crashes; rounds 2 and 3 reached 200 of 200 with none	s3-pico, one run (section 4.5)
Scale the value loss down when the values are unbounded	at $\lambda = 0.5$, with $\lvert u \rvert$ up to 74 on web text, the token loss stayed on its unigram plateau; the runs use 0.05	one observation in pretraining (section 4.2)
Pass loop dropout to one compiled graph as exit weights	with batch shapes rounded up to 32 tokens, the decision step became 4 to 5 times faster	s3-pico; the two changes were made together (section 4.3)
Choose the shape from the latency budget before training	the first of four sizes whose WebAssembly median was at most 4.0 ms; the trained file takes 2.39 ms per decision	s6-ear2-nano, a probe its pre-registration named (section 6.1)
Distil rather than train on hard labels	validation accuracy 0.8715 against 0.8437	s6-ear2-nano's pilot, arms of 12 working minutes, validation only (section 6.1)
When a teacher rewrites text, rewrite every option or none	a distractor given the rewritten description is picked on 0.8311 of the validation slates whose names are masked	s6-route-nano, a diagnosis made after its gate, on validation only (`docs/s6-route4-diagnosis.md`)
Pin the exits in the parity test	the browser at the file's stored thresholds of 0.5 against the native engine at full depth read a gap of 4.5e-2, the same answer on 199 of 200; with the same exits on both sides, 9.5e-7 on 200 of 200	s6-nano, 200 bench questions; the corrected output file is not committed (amendment 13 of `docs/s6-gen-prereg.md`)
A looped block against a stack of distinct blocks of the same size	not measured	(section 9)
Loop dropout against none	not measured	(section 9)
The numeric curriculum against none	not measured	(section 9)
One token per value against a token per digit, for accuracy	not measured; it takes a Flappy frame from 16 tokens to 9	(section 3.2)
Keeping only the teacher labels that agree with the gold answer	not measured; it kept 560,238 of 747,077 labels	(section 4.1)

fitted on held-out questions (section 5.2), the rule section 5.2 found unsafe for s2-nano; for s6-nano these were 0.5, so its rounds collected the states that early exits reach, while the file ships at full depth. s3-pico's first round fell to 5.4 Flappy pipes with 30 crashes in 30 episodes before rounds 2 and 3 recovered. No run separates the effect of these thresholds on the collected data.

- **Friendly games.** The games compute the useful features in code, so they test the decision, not perception.
- **Two graphs to keep in step.** The training and inference graphs are separate code, tied by golden tests on small random models; a difference the goldens do not exercise would go unseen.
- **One variant only made a result.** s6-sleep3-nano adds to s6-nano a fast-weight memory layer after each state block, written once per frame and read by every later frame (`docs/s6-sleep3-prereg.md`, `crates/sky-infer/src/s2/memory.rs`); the memory was used, its sleep passes added nothing measurable, and the file failed its gate (companion report [1], section 6.6).

10 Reproducibility

The code is in github.com/maxgfr/skycmd, Apache-2.0. This report describes the code at commit `ef6e995` (21 November 2026); its links point to `main`, so a reader of a later version should check out that commit. Appendix A.5 pins each published file to the commits around its run. The README's table "Reproduce a result" gives, for each s6 model, its pre-registration, its script and its result files (`README.md`); for the files of Table 1: `scripts/train-s3.sh numeric pico`, then `scripts/specialize-s3.sh pico 3 8` and `scripts/calibrate-exit.sh` for s3-pico (tutorial, chapter 21); `scripts/train-all.sh nano` (which runs `scripts/train-s6.sh`) for s6-nano; `SIZE=micro scripts/train-s6.sh` with the environment of `runs/s6-micro2/env.sh` for s6-micro; `scripts/train-turn.sh` for s6-ear2-nano; `scripts/build-route-data.sh`, `scripts/train-route.sh` and `scripts/route-eval.sh` for s6-route-nano, and `scripts/route3-read.sh` for s6-route, which trains nothing; `scripts/train-secrets2.sh` for s6-secrets-verify-pico; and `scripts/train-cast2.sh` for s6-cast2-pico. The scripts read their data from a folder next to the repository, described in `docs/data-inventory.md`. Every published file is in `site/public/models/`, listed in the manifest with its sha256. The figures of this report are written by `python3 -I scripts/paper-methods-figure.py`, and its PDF by `scripts/build-paper.sh --methods`.

Acknowledgements

Generative AI tools assisted with code and editing; the author checked every number and is responsible for the content.

References

[1] Maxime Golfier. Pre-registered gates for small models trained on one laptop: what passed, what failed, and what the readings show. Technical report, not peer-reviewed, Zenodo, 2026. Web version: https://maxgfr.github.io/skycmd/paper/.

[2] D. Almeida. Introducing System One Models & Jev. TypeSafe blog, 15 September 2026. URL https://typesafe.ai/blog/introducing-system-one-models-and-jev.

[3] cua-ai. cua-s1-forms, a form-filling option scorer (MIT). Hugging Face model card, 2026. URL https://huggingface.co/cua-ai/cua-s1-forms.

[4] Convai Innovations. Laya. Hugging Face model card, 2026. URL https://huggingface.co/convaiinnovations/laya.

[5] B. Egressy and J. Stühmer. Set-LLM: A Permutation-Invariant LLM, 2025. arXiv:2505.15433.

[6] G. Amit. Permutation-Robust Decision Modeling with Candidate-Independent Block-Causal Attention, 2026. arXiv:2610.01601.

[7] A. Jolicoeur-Martineau. Less is More: Recursive Reasoning with Tiny Networks, 2025. arXiv:2510.04871.

[8] R.-J. Zhu, Z. Wang, K. Hua, et al. Scaling Latent Reasoning via Looped Language Models, 2025. arXiv:2510.25741.

[9] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko. Revisiting Deep Learning Models for Tabular Data. In *NeurIPS*, 2021. arXiv:2106.11959.

[10] T. Zhou, D. Fu, M. Soltanolkotabi, R. Jia, and V. Sharan. FoNE: Precise Single-Token Number Embeddings via Fourier Features, 2025. arXiv:2502.09741.

[11] S. Ma, H. Wang, L. Ma, et al. The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits, 2024. arXiv:2402.17764.

[12] S. Ross, G. J. Gordon, and J. A. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *AISTATS*, 2011. arXiv:1011.0686.

[13] Y. Yue, Y. Wang, B. Kang, Y. Han, S. Wang, S. Song, J. Feng, and G. Huang. DeeR-VLA: Dynamic Inference of Multimodal Large Language Models for Efficient Robot Execution. In *NeurIPS*, 2024. arXiv:2411.02359.

[14] Samsung. CredSweeper, a credential scanner with an optional ML validator (MIT). GitHub repository, 2021. URL https://github.com/Samsung/CredSweeper.

[15] F. Boas. ML-powered feature FP Remover cuts 50% of False Positives. GitGuardian blog, June 2024. URL https://blog.gitguardian.com/fp-remover-cuts-false-positives-by-half/.

[16] mgoeckel. oscar-1-17m, a decision model on Ettin-17M. Hugging Face model card, 2026. URL https://huggingface.co/mgoeckel/oscar-1-17m.

[17] TheREZOR. TinyDecide (Apache-2.0). Hugging Face model card, 2026. URL https://huggingface.co/TheREZOR/TinyDecide.

[18] Q. Li and X. Wang. Jeva: System One Models from First Principles, 2026. arXiv:2610.18773.

[19] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser. Universal Transformers. In *ICLR*, 2019. arXiv:1807.03819.

[20] A. Graves. Adaptive Computation Time for Recurrent Neural Networks, 2016. arXiv:1603.08983.

[21] A. Banino, J. Balaguer, and C. Blundell. PonderNet: Learning to Ponder. In *8th ICML Workshop on Automated Machine Learning*, 2021. arXiv:2107.05407.

[22] A. Jeddi, M. Ciccone, and B. Taati. LoopFormer: Elastic-Depth Looped Transformers for Latent Reasoning via Shortcut Modulation. In *ICLR*, 2026. arXiv:2602.11451.

[23] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-Supervised Nets, 2014. arXiv:1409.5185.

[24] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep Networks with Stochastic Depth, 2016. arXiv:1603.09382.

[25] S. Teerapittayanon, B. McDanel, and H. T. Kung. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks, 2017. arXiv:1709.01686.

[26] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference. In *ACL*, 2020. arXiv:2004.12993.

[27] W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, and Q. Ju. FastBERT: a Self-distilling BERT with Adaptive Inference Time. In *ACL*, 2020. arXiv:2004.02178.

[28] L. Meronen, M. Trapp, A. Pilzer, L. Yang, and A. Solin. Fixing Overconfidence in Dynamic Neural Networks. In *WACV*, 2024. arXiv:2302.06359.

[29] T. Schuster, A. Fisch, T. Jaakkola, and R. Barzilay. Consistent Accelerated Inference via Confident Adaptive Transformers. In *EMNLP*, 2021. arXiv:2104.08803.

[30] A. N. Angelopoulos, S. Bates, A. Fisch, L. Lei, and T. Schuster. Conformal Risk Control. In *ICLR*, 2024. arXiv:2208.02814.

[31] T. Schuster, A. Fisch, J. Gupta, M. Dehghani, D. Bahri, V. Q. Tran, Y. Tay, and D. Metzler. Confident Adaptive Language Modeling. In *NeurIPS*, 2022. arXiv:2207.07061.

[32] M. Jazbec, A. Timans, T. Hadži Veljković, K. Sakmann, D. Zhang, C. A. Naesseth, and E. Nalisnick. Fast yet Safe: Early-Exiting with Risk Control. In *NeurIPS*, 2024. arXiv:2405.20915.

[33] S. Khazem. SAFE-KD: Risk-Controlled Early-Exit Distillation for Vision Backbones, 2026. arXiv:2602.03043.

[34] Y. Bengio, N. Léonard, and A. Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation, 2013. arXiv:1308.3432.

[35] J. Nielsen and P. Schneider-Kamp. BitNet b1.58 Reloaded: State-of-the-art Performance Also on Smaller Networks. In *DeLTA, Springer CCIS 2172*, 2024. arXiv:2407.09527.

[36] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks, 2016. arXiv:1603.05279.

[37] F. Li, B. Liu, X. Wang, B. Zhang, and J. Yan. Ternary Weight Networks, 2016. arXiv:1605.04711.

[38] B. Warner, A. Chaffin, B. Clavié, O. Weller, et al. Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference, 2024. arXiv:2412.13663.

[39] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding, 2021. arXiv:2104.09864.

[40] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On Calibration of Modern Neural Networks. In *ICML*, 2017. arXiv:1706.04599.

[41] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. NIPS 2014 Deep Learning Workshop, 2015. arXiv:1503.02531.

[42] I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2019. arXiv:1711.05101.

[43] K. Jordan, Y. Jin, V. Boza, J. You, F. Cesista, L. Newhouse, and J. Bernstein. Muon: An optimizer for hidden layers in neural networks. Blog post, December 2024. URL https://kellerjordan.github.io/posts/muon/.

[44] K. Wen, D. Hall, T. Ma, and P. Liang. Fantastic Pretraining Optimizers and Where to Find Them, 2025. arXiv:2509.02046.

[45] M. Akhtar, H. N. Nounou, M. Mansouri, and M. N. Nounou. When Does Muon Actually Win? A Paired Data-, Time-, and Tuning-Budget Benchmark, 2026. arXiv:2610.21555.

[46] K. Helwani, H. Do, J. Luan, and S. Srinivasan. A Hierarchical End-of-Turn Model with Primary Speaker Segmentation for Real-Time Conversational AI, 2026. arXiv:2603.13379.

[47] aurelio-labs. semantic-router (MIT). GitHub repository, 2023. URL https://github.com/aurelio-labs/semantic-router.

[48] S. Robertson and H. Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, 2009. doi:10.1561/1500000019.

[49] J. Yun, S. Choi, Y. Lee, O. Sokol, W. Shim, A. Melkonyan, and D. Kuzmenko. Project CredData: A Dataset of Credentials for Research. Samsung, GitHub repository (Apache-2.0), 2021. URL https://github.com/Samsung/CredData.

[50] T. Zhan, H. Zhang, and Y. He. Fracast-0: Fractal Weight Sharing for a Time Series Foundation Model with Only 85K Parameters, 2026. arXiv:2609.32209.

[51] Tracel AI. Burn, a deep learning framework in Rust. GitHub repository, 2022. URL https://github.com/tracel-ai/burn.

[52] Y. Gorishniy, I. Rubachev, and A. Babenko. On Embeddings for Numerical Features in Tabular Deep Learning. In *NeurIPS*, 2022. arXiv:2203.05556.

[53] S. Golkar, M. Pettee, M. Eickenberg, A. Bietti, M. Cranmer, G. Krawezik, F. Lanusse, M. McCabe, et al. xVal: A Continuous Numerical Tokenization for Scientific Language Models, 2023. arXiv:2310.02989.

[54] Allen Institute for AI. Dolma 3 (`dolma3_mix-150B-1025`), the pretraining data of OLMo 3 (ODC-BY-1.0). Hugging Face data set, 2025. URL https://huggingface.co/datasets/allenai/dolma3_mix-150B-1025.

[55] LLM360 (now IFM). TxT360 (ODC-BY-1.0). Hugging Face data set, 2024. URL https://huggingface.co/datasets/IFM/TxT360.

[56] zai-org. GLM-5.3-Flash, an open-weight model (MIT). Hugging Face model card, 2026. URL https://huggingface.co/zai-org/GLM-5.3-Flash.

[57] S. Ross and J. A. Bagnell. Efficient Reductions for Imitation Learning. In *AISTATS, PMLR 9*, pages 661–668, 2010. https://proceedings.mlr.press/v9/ross10a.html.

[58] H. Ross, A. S. Mahabaleshwarkar, and Y. Suhara. When2Call: When (not) to Call Tools. In *NAACL*, 2025. https://aclanthology.org/2025.naacl-long.174/; data (CC BY 4.0): https://huggingface.co/datasets/nvidia/When2Call.

[59] P. Boersma. Accurate Short-Term Analysis of the Fundamental Frequency and the Harmonics-to-Noise Ratio of a Sampled Sound. Institute of Phonetic Sciences, University of Amsterdam, Proceedings 17, 97–110, 1993. URL https://www.fon.hum.uva.nl/paul/papers/Proceedings_1993.pdf.

[60] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever. Robust Speech Recognition via Large-Scale Weak Supervision, 2022. arXiv:2212.04356.

[61] Pipecat. Smart Turn v3.2, an open end-of-turn model (BSD-2-Clause). GitHub repository, 2026. URL https://github.com/pipecat-ai/smart-turn.

[62] X. H. Lù. BM25S: Orders of magnitude faster lexical search via eager sparse scoring, 2024. arXiv:2407.03618.

[63] Minish Lab. potion-base-2M, a Model2Vec static embedding (MIT). Hugging Face model card, 2024. URL https://huggingface.co/minishlab/potion-base-2M.

[64] S. Tulkens and T. van Dongen. Model2Vec: Fast State-of-the-Art Static Embeddings (MIT). Zenodo, 2024. Code: https://github.com/MinishLab/model2vec.

[65] SAP. Credential Digger (Apache-2.0). GitHub repository, now archived, 2020. URL https://github.com/SAP-archive/credential-digger. Method in S. Lounici, M. Rosa, C. M. Negri, S. Trabelsi, M. Önen, *Optimizing Leak Detection in Open-Source Platforms with Machine Learning Techniques*, ICISSP 2021.

[66] M. Golfier. secretgate, a local secrets firewall for coding agents that redacts credentials (MIT). GitHub repository, 2026. URL https://github.com/maxgfr/secretgate.

[67] Gitleaks. Gitleaks, a secret scanner (MIT). GitHub repository, 2018. URL https://github.com/gitleaks/gitleaks.

[68] RAWS Labs. TinyCast, a 146,505-parameter forecaster (Apache-2.0). Hugging Face model card, 2026. URL https://huggingface.co/raws-labs/tinycast.

[69] A. Steinhauser. TinyCast: Probabilistic Zero-Shot Forecasting with Computed Periodicity, 2026. arXiv:2608.15767.

[70] T. Aksu, G. Woo, J. Liu, X. Liu, C. Liu, S. Savarese, C. Xiong, and D. Sahoo. GIFT-Eval: A Benchmark For General Time Series Forecasting Model Evaluation, 2024. arXiv:2410.10393; data: https://huggingface.co/datasets/Salesforce/GiftEval.

[71] J. Wei, S. Cao, T. Cao, L. Ma, L. Wang, Y. Zhang, and M. Yang. T-MAC: CPU Renaissance via Table Lookup for Low-Bit LLM Deployment on Edge. In *EuroSys*, 2025. arXiv:2407.00088.

[72] J. Wang, H. Zhou, T. Song, S. Cao, Y. Xia, T. Cao, J. Wei, S. Ma, et al. Bitnet.cpp: Efficient Edge Inference for Ternary LLMs, 2025. arXiv:2502.11880.

A Exact specification

This appendix holds the details a reimplementation needs and a first reading does not.

A.1 Notation

Each symbol keeps one meaning throughout; the indices i and j are local (options, tokens, columns), and code names are written in `code`.

A.2 Truncation and temperature lookup

Truncation. Each option is cut to its marker and 48 tokens; if the options leave less than 16 tokens of the head window, every option is cut to max(4, (max_head − 16) / n) tokens, and the question keeps at least 8 (`crates/sky-infer/src/s2/prompt.rs`).

Temperature lookup. The engine looks up the temperature of (exit, type, bucket), then (exit, type), then (type, bucket), then the type, and uses 1 when none is stored; the training code has only the first two levels (`crates/sky-infer/src/s2/mod.rs`, `crates/sky-train/src/s2/config.rs`).

Notation.

symbol	meaning	symbol	meaning
d	model width	n	number of options of a question
B, L	distinct state blocks, loops over them	t, $b(n)$	question type, bucket of n
l	loop index, from 0	T	a softmax temperature
e, N_x	exit index, number of exits (the last exit is N_x)	τ_t	exit threshold of type t
$u(v)$, $\varphi(v)$	signed logarithm of a value v, its 17 features	γ_o	scale of output row o of a ternary matrix
ω_k	frequency k of φ, $k = 0, \ldots, 7$	q	fraction of a run trained ternary
W_{tok}, χ_i	token embedding table, value flag of token i	$\mathcal{L}$, w_e	decision loss, weight of exit e
H_e, m_i	state after exit e, marker vector of option i	λ	weight of the value loss in pretraining
s_i, $p_i^{(e)}$	logit and probability of option i at exit e	β_1, β_2	AdamW moment decays
R, r	DAgger rounds, round index	μ_r	oracle share of round r
n_{ep}	episodes per game in Algorithm 2	$A_g(\tau)$	metrics of game g at threshold τ
n_{dec}, $\hat{R}(\tau)$, α	decisions, empirical risk and risk level of the conformal rule	η	duty cycle of the brake
ν	quantile level of the forecaster	δ	a decision threshold of a task-specific system
κ	fusion exponent of the router	c_{none}, T_{emb}	constant of *none*, temperature of the router's embedding part

A.3 What the published decision files do not use

Some options of the code are in no published decision file. *Temporal carry* adds to each frame's state embedding a learned gate, starting at 0, times the LayerNorm of the previous frame's final latent; on s3-pico it did not reduce the depth used and was not shipped (`BENCH.md`, section "Temporal recursion"). *Parallax* and *exclusive self-attention* change the encoder's attention; the ablation of section 4.4 adopted neither, so every trained s6 file has plain softmax attention, though some configuration files still say `"parallax": true` before the recipe script overwrites it (`scripts/s6-adopt.py`). The fast-weight memory of s6-sleep3-nano failed its gate (section 9).

A.4 The two recipes of Muon

Recipe 1, the code of commit `e678f80` (12 October 2026) that added s3-pico, applies Muon with Nesterov momentum 0.95 (warmed from 0.85 over 300 steps) to the encoders' hidden matrices and AdamW to everything else: an L2 term of 0.01 is added to the gradient, five Newton-Schulz steps with the coefficients (3.4445, −4.7750, 2.0315) orthogonalise the update, and it is scaled by $\sqrt{\max(1, n_{\text{out}}/n_{\text{in}})}$ for an $n_{\text{out}} \times n_{\text{in}}$ matrix. Its peak rates are 3e-3 for AdamW and 0.02 for Muon in pretraining, 1e-3 and 0.01 in decision training and the DAgger rounds, with no width factor. **Recipe 2**, from commit `9191533` (24 October 2026), runs the five steps and then two with (15/8, −10/8, 3/8), scales the update by $0.2 \cdot \sqrt{\max(n_{\text{out}}, n_{\text{in}})}$ and shares AdamW's rate and decoupled decay (`docs/SPEC-S6.md` §2); the code at the commit this report describes keeps only recipe 2 (`crates/sky-train/src/optim.rs`).

A.5 Which commit trained which file

The run folders do not record the commit of the binary that trained a file, so each file is pinned here to the two commits around its run, from the companion report's Table 2: the run used code committed between the commit that froze its pre-registration and the commit that first records its reading, and possibly changes not yet committed. s6-nano: `30e2530` to `dec164e`; s6-micro v2: `aca7f91` to `d3394cf`; s6-ear2-nano: `8f70df8` to `2519daf`; s6-route-nano: `e27a2ab` to `25c5c2d`; s6-secrets-verify-pico: `f842658` to `bc7a1d6`; s6-cast2-pico: `44f1cf2` to `bdea512`. These six use AdamW

only, so the recipe of Muon in the code does not affect them; later commits to their training scripts changed paths and error handling, not settings. s3-pico, which had no pre-registration, entered the repository with its scripts in commit `e678f80` (12 October 2026), trained with Muon's recipe 1 (section 4.4). Recipe 2 replaced it in `9191533` (24 October 2026), and the code at `ef6e995` has no recipe 1, so `scripts/train-s3.sh numeric pico` at that commit trains another recipe than the published file's: to retrain s3-pico as published, check out `e678f80`.

How to cite

```
@misc{golfier2026skycmdmethods,
  author       = {Golfier, Maxime},
  title        = {Looped ternary encoders for typed decisions:
                  a recipe from laptop to browser},
  year         = {2026},
  publisher    = {Zenodo},
  doi          = {10.5281/zenodo.24420817},
  howpublished = {\url{https://maxgfr.github.io/skycmd/paper/methods/}},
  note         = {Technical report, not peer-reviewed}
}
```

This report is archived on Zenodo with the DOI 10.5281/zenodo.24420817. The companion report's citation is in `CITATION.cff`. The report is updated when a method's description changes; the date at the top is that of its last change.