Stylus: Learning to Draw Like a Hand by Inverting a Renderer

Luka Löhr
Independent Researcher

Abstract

Diagram tools and generative image models produce finished pictures, not the process of drawing them. We study the inverse problem: given a raster sketch, recover a time-ordered sequence of pen movements that a person could plausibly have made to draw it, including where and when to write handwritten labels. We present Stylus-0.9B, a ∼0.93B-parameter *motor model* that reads a sketch with a SigLIP vision encoder and emits pen events (pen-up moves and cubic Bézier segments with quantised durations) with a Qwen2.5-0.5B decoder, one sequence position per event. A lightweight autoregressive field decoder factorises each event over its opcode, coordinates and duration without additional transformer passes. The model is trained without paired image–trajectory annotations by *render-and-invert* supervision: human and procedurally composed pen trajectories are rendered, degraded by a stochastic operator that mimics imperfect sketches, and the model learns to reconstruct the original events. Training data combine timed Quick, Draw! sketches and MathWriting handwriting filtered for kinematic smoothness with 150,000 procedurally generated whiteboard scenes built from human glyphs and icons. A pointer prior over image patches, added after close visual checks showed that the model drew plausible but unrelated sketches, made it follow the input. On held-out test sets, Stylus orders strokes much more like humans than a skeletonisation baseline (Kendall τ 0.90 vs. 0.69 on handwritten formulas, 0.75 vs. 0.26 on diagrams) and moves with a jerk closer to human on formulas and doodles. It is, however, far less precise than tracing the input (Chamfer 7.9–30.6 vs. below 1 canvas unit), fragments strokes, often writes illegible letters, and fails on dense calligraphy. Combined with a frozen LLM planner and a LoRA-adapted SANA sketch generator, Stylus turns text requests into animated, hand-drawn diagrams, reliably for simple box-and-arrow layouts and not yet for drawings of objects. All models were trained on one rented GPU in 5.4 h. This is a proof of concept: it shows that such a model can be trained from public data and that it learns stroke order and timing; finishing it to precise geometry and legible handwriting would, we estimate, take weeks of training on high-end multi-GPU hardware.

1 Introduction

Explanations at a whiteboard are drawn, not displayed. A box appears, a label is written into it, an arrow is pulled towards the next box. The order and timing of these strokes carry information: they tell the viewer what is being introduced, what depends on what, and where to look. Programmatic diagram tools (graph layout engines, slide software, vector editors) produce clean final layouts but no drawing process; animating them typically means revealing finished paths at constant speed, which looks mechanical. Text-to-image models produce convincing hand-drawn *pictures*, but a picture is a raster without strokes, order or timing, and current diffusion models cannot reliably render legible text.

We consider the problem of turning a raster sketch into a plausible human drawing process. Given an image and, optionally, a list of text strings with target boxes, the system must output a time-ordered sequence of pen events: where the pen travels while lifted, which curves it traces while down, how long each movement takes, and when it writes each label. We call a model for this task a *motor model*, in analogy to the motor stage that executes a visual plan.

The central obstacle is data. Large collections of timed human pen trajectories exist for isolated doodles [14] and handwriting [1, 8], but there is no corpus of whiteboard diagrams with recorded strokes, and none paired with the imperfect raster images that a generative model would produce. We side-step pairing entirely: if we have trajectories, we can render them. Stylus is trained by *render-and-invert* supervision. Ground-truth pen events are rendered by a stochastic degradation operator that varies stroke width and ink tone, blurs, warps and adds noise, and replaces some handwritten labels by blank grey banners; the model learns to recover the original events from the degraded image. Because the degradation is applied on the fly, every epoch sees new images of the same trajectories.

To cover the multi-object layouts of real diagrams, we compose synthetic whiteboard scenes from human ink: Quick, Draw! icons, handwriting assembled from human-written MathWriting glyphs, and geometric primitives (boxes, cylinders, circles, diamonds, arrows) whose kinematics follow the minimum-jerk model [7]. Scenes are drawn in an order that follows common sketching habits (a container before its label, an arrow after both of its end nodes). Human sources are filtered for kinematic smoothness using the dimensionless jerk [12].

Scope. This paper is a proof of concept. Its aim is to show that a motor model of this kind can be built and trained at all: without paired data, from public ink corpora, on a single rented GPU in a few hours. The model we report learns the parts of the problem that concern the drawing *process*, stroke order and timing, but its geometry is coarse and its handwriting is often illegible. It saw less than one pass over its training data; we expect that a finished model needs one to two orders of magnitude more training, i.e. weeks on high-end multi-GPU hardware, together with the changes discussed in §6.

Contributions.

- An **event representation** for timed pen trajectories with pen-up moves, cubic Bézier pen-down segments, log-quantised durations and explicit text-anchor delimiters, together with an encoder that fits human ink to it (§3.1).
- **Stylus-0.9B**, a vision-language motor model with one sequence position per event and an autoregressive *field decoder* that factorises each event over opcode and fields using small per-field heads instead of extra transformer passes; coordinates and durations are trained with Gaussian-smoothed targets (§3.2).
- **Render-and-invert training**: a self-supervised degradation operator that turns any trajectory corpus into image-to-trajectory training pairs, including a banner mechanism that teaches the model to write specified text into blank regions (§3.3).
- A **data pipeline** that combines jerk-filtered human ink with a procedural whiteboard-scene grammar with human-like drawing order and minimum-jerk primitives (§3.4).
- A **three-stage text-to-drawing pipeline**: a frozen LLM planner, a LoRA-adapted SANA sketch generator that leaves grey banners for text, and the motor model, which writes the labels into the detected banners (§3.5).
- An **evaluation** of fidelity, stroke order and kinematics against a skeletonisation-and-tracing baseline (§5). Code, trained weights (non-commercial research use) and the data-building scripts accompany this preprint (`https://github.com/luka-loehr/stylus-0.9B`).

2 Related Work

Sequential sketch generation. Sketch-RNN [11] models Quick, Draw! drawings as sequences of pen offsets and pen states with a recurrent VAE and mixture-density outputs, following the handwriting synthesis model of Graves [10]. Sketchformer [23] learns transformer representations of the same stroke-3 format for recognition and retrieval. SketchKnitter [27] generates vectorised sketches with a diffusion process over stroke points and pen states. These models generate sketches unconditionally or from a class; they neither read a given image nor model absolute timing. Stylus uses the same human data, but conditions on a raster and predicts curve segments with durations.

Vector graphics generation. DeepSVG [3] uses a hierarchical transformer over SVG commands. IconShop [28] tokenises SVG paths and text and trains an autoregressive transformer for text-guided icon synthesis, and StarVector [24] couples an image encoder with a code LLM to emit SVG code from images and text. These systems target the final vector *shape*: the order of SVG primitives is a file-format artefact, and there is no notion of pen-up travel or movement time. Optimisation-based approaches render vector primitives through a differentiable rasteriser [17]; CLIPasso [26] optimises Bézier strokes to abstract a photo under a CLIP-based loss. Stylus also inverts a renderer, but by amortised learning over a stochastic, non-differentiable degradation operator, and it outputs a drawing process rather than a static set of strokes.

Vectorisation and inverse graphics. Classical vectorisation thins a binary image [31], traces the skeleton and fits curves [25]; learned variants predict primitives for technical drawings [4]. Ellis et al. [5] infer graphics programs from hand-drawn images. Stroke recovery from offline handwriting reconstructs pen trajectories from images of text [2]. Our skeletonisation baseline (§4.1) follows the classical route; Stylus differs in that it learns stroke segmentation, order and timing from human data and handles mixed drawings and text.

Handwriting and movement kinematics. Graves [10] synthesises online handwriting with a recurrent mixture-density network. Models of human movement describe point-to-point reaching as minimising integrated squared jerk [7], and handwriting strokes as superpositions of lognormal velocity impulses [21]. Hogan and Sternad [12] show that only dimensionless jerk measures are robust to movement duration and amplitude. We use the minimum-jerk profile to give synthetic primitives human-like timing and a dimensionless jerk score both to filter data and as an evaluation metric.

Vision-language backbones and generative image models. Stylus pairs a SigLIP [30] vision encoder with a Qwen2.5 [22] decoder through an MLP projector, as in LLaVA-style models [19], but replaces the text vocabulary by structured event embeddings and heads. Predicting several quantities from one hidden state with light heads relates to multi-token prediction [9]; our heads are conditioned on each other sequentially within one event. The sketch generator in our pipeline is SANA [29], adapted with LoRA [13] under a flow matching objective [18, 20] with logit-normal timestep sampling [6].

3 Method

Stylus maps an image I and an optional list of text anchors $\mathcal{A} = \{(a, s_a, b_a)\}$ (anchor id a, string s_a, box b_a) to a sequence of pen events $e_{1:L}$. We first describe the event representation (§3.1), then the model (§3.2), the render-and-invert training signal (§3.3), the training data (§3.4) and the full text-to-drawing pipeline (§3.5).

Table 1: Event vocabulary (`stylus/events.py`). Each event is stored as nine integers $(o, x_1, y_1, x_2, y_2, x_3, y_3, \delta, a)$; unused fields are -1 and are neither embedded nor predicted.

Opcode	Fields	Meaning
`PAD`	–	padding, never predicted
`BOS`	–	start of drawing
`EOS`	–	end of drawing
`MOVE`	x_3, y_3, δ	pen-up travel to (x_3, y_3) after pen-up time δ
`CURVE`	$x_1, y_1, x_2, y_2, x_3, y_3, \delta$	pen-down cubic Bézier from the current position, duration δ
`TEXT_START`	a	following events write anchor a
`TEXT_END`	–	end of the current anchor

Field sizes: coordinates 1000 bins per axis; δ: 40 log-spaced bins, 5–800 ms; a: 16 ids.

3.1 Event representation

A drawing lives on a 1000×1000 canvas. An event is a tuple $e = (o, \mathbf{f})$ of an opcode $o \in \{\texttt{PAD}, \texttt{BOS}, \texttt{EOS}, \texttt{MOVE}, \texttt{CURVE}, \texttt{TEXT_START}, \texttt{TEXT_END}\}$ and a vector of integer fields $\mathbf{f} = (x_1, y_1, x_2, y_2, x_3, y_3, \delta, a)$, of which only the subset $\mathcal{F}(o)$ used by the opcode is defined (Table 1):

- `MOVE`(x_3, y_3, δ): pen-up travel to (x_3, y_3); δ is the pen-up time since the previous pen-down segment ended.
- `CURVE`$(x_1, y_1, x_2, y_2, x_3, y_3, \delta)$: a pen-down cubic Bézier segment from the current pen position P_0 with control points $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ and end point $P_3 = (x_3, y_3)$, traversed in time δ.
- `TEXT_START`(a) / `TEXT_END`: delimit the events that write text anchor $a \in \{0, \ldots, 15\}$.

Coordinates are rounded to 1000 bins per axis. Durations are quantised on a logarithmic grid of 40 bins between $\delta_{\min} = 5$ ms and $\delta_{\max} = 800$ ms,

$$q(\delta) = \operatorname{round}\left(39 \cdot \frac{\log \operatorname{clip}(\delta, \delta_{\min}, \delta_{\max}) - \log \delta_{\min}}{\log \delta_{\max} - \log \delta_{\min}}\right), \tag{1}$$

so that consecutive bins differ by a factor of $(160)^{1/39} \approx 1.14$. Every drawing is wrapped in `BOS` and `EOS`.

Encoding human ink. A timed stroke is a polyline $\{(x_i, y_i, t_i)\}$. After removing duplicate points and enforcing non-decreasing time, each stroke is split at sharp corners (turning angle above 55° on a Ramer–Douglas–Peucker simplification with tolerance 2.5 canvas units). Each piece is fitted with piecewise cubic Bézier curves by recursive least squares with Newton reparametrisation in the style of Schneider [25], splitting at the point of maximum error until it falls below 2.0 canvas units. A stroke becomes one `MOVE` followed by one `CURVE` per fitted segment, whose duration is the elapsed time between the segment's end samples (at least 5 ms). Decoding samples each Bézier segment at a spacing of about 2 canvas units and interpolates time linearly in the curve parameter.

3.2 Model

Architecture. The image is rendered or resized to 384×384 grey levels and encoded by SigLIP So400m/14 [30] without its pooling head, giving 729 patch features in $\mathbb{R}^{1152}$. A two-layer MLP ($1152 \to 896 \to 896$, GELU) projects them to the width of the Qwen2.5-0.5B [22] decoder (24 layers, $d = 896$), which we use without its language-model head. The text prompt (§B), which

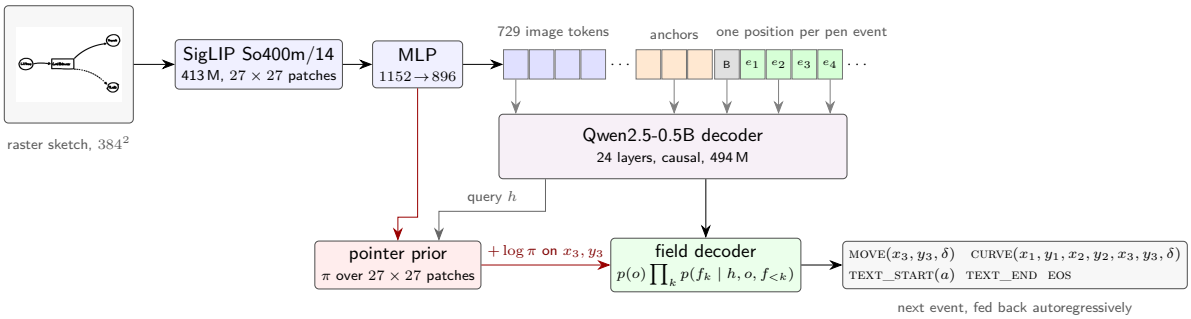


Figure 1: Stylus motor model. A SigLIP encoder turns the raster into 27×27 patch tokens, which are projected and prepended to the text-anchor prompt and the pen-event sequence. A Qwen2.5-0.5B decoder reads the sequence causally; at every position the field decoder predicts the next event part by part, and a pointer over the patch grid adds a spatial log-prior to the end point (x_3, y_3).

Table 2: Parameter counts of Stylus-0.9B (`Stylus.num_params()`). Counted on the instantiated model.

Component	Parameters
Vision encoder (SigLIP So400m/14, no head)	412,987,248
Projector (MLP $1152 \to 896 \to 896$)	1,836,800
Backbone (Qwen2.5-0.5B, no LM head)	494,032,768
Event embeddings (Eq. 2)	5,432,448
Event head (opcode head + field decoder)	19,008,687
Pointer prior (Eq. 6)	461,056
Total	933,759,007

lists the anchors with their boxes and strings, is embedded with the Qwen token embeddings. Events occupy *one sequence position each*; the input embedding of event $e_t = (o_t, \mathbf{f}_t)$ is

$$\mathbf{x}_t = E_{\text{op}}[o_t] + \sum_{k \in \mathcal{F}(o_t)} E_k[f_{t,k}], \tag{2}$$

with one embedding table per field. The backbone runs causally over the concatenation

$$[\,\text{image tokens};\ \text{prompt tokens};\ \mathbf{x}_0, \ldots, \mathbf{x}_{L-1}\,],$$

and the hidden state $\mathbf{h}_t$ at event position t predicts event e_{t+1}. Table 2 lists parameter counts.

Field decoder. One event contains up to eight interdependent values. Predicting them independently from $\mathbf{h}_t$ would ignore, for example, that control points depend on the end point. We factorise

$$p(e \mid \mathbf{h}) = p(o \mid \mathbf{h}) \prod_{k \in \pi(o)} p(f_k \mid \mathbf{h}, o, \mathbf{f}_{\pi<k}), \tag{3}$$

where π is the fixed order $(x_3, y_3, x_1, y_1, x_2, y_2, \delta, a)$ restricted to $\mathcal{F}(o)$: the end point first, the control points conditioned on it, the duration last. The conditionals are computed without further transformer passes. A context vector is initialised with an opcode embedding and accumulates one embedding per decoded field,

$$\mathbf{c}^{(0)} = \mathbf{h} + U_{\text{op}}[o], \qquad \mathbf{c}^{(j+1)} = \mathbf{c}^{(j)} + U_{\pi_j}[f_{\pi_j}], \tag{4}$$

and the logits of the j-th field are produced by a field-specific MLP with hidden width 1024,

$$\boldsymbol{\ell}_{\pi_j} = W^{\text{out}}_{\pi_j} \,\text{GELU}(W^{\text{up}}_{\pi_j} \,\text{LN}_{\pi_j}(\mathbf{c}^{(j)})). \tag{5}$$

Fields not used by the opcode are skipped and do not update $\mathbf{c}$. The opcode logits are a linear function of $\mathbf{h}$. During training, ground-truth opcodes and fields are fed in (teacher forcing) and all heads run in parallel over all positions.

Pointer prior for end points. A linear head over 1000 coordinate bins has to learn where things are in the image only through the backbone's hidden state, and in our first training phase the end-point losses plateaued while the sequence prior kept improving (§5). We therefore add a spatial pointer over the $G \times G$ grid of vision patches ($G = 27$). With projected vision tokens $\mathbf{v}_p$ and the opcode-augmented context $\mathbf{c}^{(0)}$,

$$\pi_p = \text{softmax}_p\left(\frac{(W_q \,\text{LN}(\mathbf{c}^{(0)}))^\top W_k \mathbf{v}_p}{\sqrt{d_p}}\right), \qquad d_p = 256, \tag{6}$$

is a distribution over patches (rows r, columns c). It enters the end-point logits as a log-prior: for x_3 we add $\log \sum_r \pi_{r,c(x)} - \log n_{c(x)}$, where $c(x)$ is the column containing bin x and n_c the number of bins per column, and for y_3 we add the analogous row term of $\pi_{\cdot,c(x_3)}$ renormalised within the column of the already chosen x_3. The field MLP then only has to resolve the position inside a patch. The pointer adds 0.46 M parameters and is initialised so that its prior is close to uniform, which lets it be added to a partially trained model without disrupting it.

Smoothed targets. Coordinate and duration bins are ordinal, so an off-by-one prediction should cost less than a distant one. For a target bin y we use a discretised Gaussian target $\tilde{q}_\sigma(j \mid y) \propto \exp(-(j-y)^2/2\sigma^2)$ for $|j-y| \le \lceil 3\sigma \rceil$ (renormalised at the range boundaries), and the loss

$$\mathcal{L} = \text{CE}(o) + \sum_k \mathbb{E}_{t:\, k \in \mathcal{F}(o_t)}\Big[-\sum_j \tilde{q}_{\sigma_k}(j \mid f_{t,k}) \log p(j \mid \mathbf{h}_t, o_t, \mathbf{f}_{t,\pi<k})\Big], \tag{7}$$

with $\sigma = 1.5$ bins for coordinates, $\sigma = 1$ for durations and a one-hot target for the anchor id. Each field term is averaged over the events in the batch that use the field.

Decoding. At inference we sample events autoregressively with a key–value cache, with temperature 0.6–0.7 and top-$k = 20$ for fields. A grammar mask forbids `PAD` and `BOS`, `TEXT_START` inside a text span, `TEXT_END` outside one, and anchor ids that do not exist or have already been written. Generation stops at `EOS` or after 1500 events.

Ink budget (optional). A recurring failure of the sampler is a degenerate loop: the model keeps placing short curves at one location and fills it with dense scribbles. As an optional decoding constraint we count the `CURVE` end points that fall into each of the 27×27 patches and, once a patch holds c end points, set its pointer score to $-\infty$ for subsequent `CURVE` events, so that their end points can no longer land there (`MOVE` is not restricted). We set $c = 24$, the largest number of end points per patch in any human drawing of our four test sets (the per-drawing maximum has a median of 2 on Quick, Draw! and MathWriting and 10 on scenes), so the budget never binds on human-like density. The budget is a decoding heuristic, not part of the model; all results are reported without it unless stated.

3.3 Render-and-invert supervision

Let $e_{1:L}$ be a ground-truth event sequence with anchors $\mathcal{A}$. A stochastic degradation operator $D(\cdot\,; \xi)$ with random parameters ξ produces a training image $I = D(e_{1:L}, \mathcal{A}; \xi)$, and the model



Figure 2: Render-and-invert training pairs from the four sources (held-out samples). Odd columns: the target event sequence Y, coloured by drawing time from dark (early) to yellow (late). Even columns: the degraded raster $\mathcal{D}(Y)$ the model receives (random stroke width, ink tone, elastic warp, blur and noise). Text anchors are replaced by grey banners with probability 0.5; then, as in the MathWriting example on the right, the model has to write the anchor string without seeing the ink.

Table 3: Parameters of the degradation operator D (`stylus/data/render.py`), applied at 384 px during motor-model training.

Step	Parameters	Probability
Banner per anchor	grey $200 + \mathcal{U}\{-15, \dots, 14\}$, pad 6 units, text hidden	0.5
Stroke width	$\mathcal{U}(0.8, 1.8)$ px, anti-aliased[†]	1
Ink tone	$\mathcal{U}\{0, \dots, 69\}$ (0 = black)	1
Elastic warp	amplitude $\mathcal{U}(0.5, 2.5)$ px, smoothing $\sigma \sim \mathcal{U}(6, 14)$ px	0.6
Gaussian blur	$\sigma \sim \mathcal{U}(0.4, 1.4)$ px	0.7
Additive noise	std $\mathcal{U}(2, 8)$ grey levels	0.5

[†]Second training phase; the first phase used $\mathcal{U}(1.2, 3.6)$ px (§3.3).

is trained to maximise $\log p_\theta(e_{1:L} \mid I, \mathcal{A})$. D is a composition of the following steps (parameters in Table 3):

1. **Banners.** Each anchor is independently replaced, with probability 0.5, by a filled grey rectangle (grey level 200 ± 15, padded by 6 canvas units) drawn under the other ink; the handwriting of that anchor is not rendered. The target sequence still contains the text events, so the model must write the string given in the prompt into the banner. Anchors that are not hidden remain visible handwriting that the model learns to trace.

2. **Ink.** The remaining strokes are rendered anti-aliased with a random width (0.8–1.8 px at 384 px) and ink tone. In the first training phase we used 1.2–3.6 px, which merged 9-pixel-high handwriting into solid blobs; thinner strokes keep letters legible at the encoder resolution.

3. **Elastic warp** (probability 0.6): a displacement field of Gaussian-smoothed uniform noise.

4. **Blur** (probability 0.7) and **sensor noise** (probability 0.5).

Images are regenerated on the fly for each sample, so the model never sees the same image twice. The operator is designed to cover the imperfections of the Stage 2 sketcher (§3.5): slightly wobbly, blurred lines of varying weight and grey banners in place of text.

3.4 Training data

Kinematic smoothness score. Human ink varies in quality; hurried or jittery drawings are poor targets for a model meant to look natural. We rank drawings by the dimensionless squared jerk [12]. For a stroke $\mathbf{p}(t)$ of duration T and path length Λ, resampled at 100 Hz,

$$\mathrm{DLJ}(\mathbf{p}) = \frac{T^5}{\Lambda^2} \int_0^T \left\| \dddot{\mathbf{p}}(t) \right\|^2 dt, \qquad J = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{p} \in \mathcal{S}} \log\bigl(1 + \mathrm{DLJ}(\mathbf{p})\bigr), \tag{8}$$

where derivatives are finite differences and $\mathcal{S}$ contains the strokes longer than 50 ms. Lower J is smoother; the offset of one keeps degenerate constant-velocity strokes (zero jerk) finite. We normalise by path length rather than by movement amplitude, which keeps the score defined for closed strokes.

Sources.

- **Quick, Draw!** [14] raw data with timestamps. Per category we read the first 5000 drawings, keep recognised drawings with 2–30 strokes and a total duration of 0.3–25 s, and keep the smoothest 45% by J. The 60 smoothest training drawings per category form the *icon bank* used by the scene generator.

- **MathWriting** [8]. Human-written training expressions with at most 60 strokes are ranked by J; the smoothest 30% (at most 60,000) become text samples with a single anchor whose string is the normalised label. A *glyph bank* of human-written single characters is built from the symbol annotations (`symbols.jsonl`, strokes cut out of training inks) and the standalone single-symbol inks. Per character we drop glyphs whose log aspect ratio deviates from the character's median by more than 0.45 or that use more than the median number of strokes plus one, and keep the smoothest 60% of the rest (at most 150); this yields 2,339 glyphs for 80 characters.
- **Calliar** [1]: Arabic calligraphy strokes without timestamps, timed at constant speed with 120 ms pen-up gaps, used as plain drawings without anchors.
- **Synthetic whiteboard scenes** (below).

Each sample is placed into a random box on the canvas with its aspect ratio preserved, encoded as in §3.1, and discarded if it exceeds 1200 events. A 1% test split is defined by a hash of the sample key.

Scene grammar. A scene generator composes multi-object diagrams from four kinds of items: *containers* (box, cylinder, circle, diamond), *icons* (from the icon bank), *labels* (handwriting) and *arrows* (straight or curved, optionally dashed, with one- or two-stroke heads, optionally carrying a small icon). Four layouts are sampled: a left-to-right or top-to-bottom *pipeline* of 2–4 nodes ($p = 0.35$), a *fan-out* from a source through a hub to 2–4 targets (0.30), a *tree* of 2–3 levels (0.20), and an *annotated* large icon with 1–3 pointer labels (0.15). Labels are drawn from a vocabulary of system-design and ML terms or from the icon category name. Each scene also yields a caption describing the layout and quoting every label, which is used to train the Stage 2 sketcher.

Drawing order follows habits of whiteboard sketching: nodes are visited in layout order; a node's container is drawn first, then its label, then its icon (label first with probability 0.2); an arrow is drawn after both of its end nodes exist.

Handwriting is composed from the glyph bank: each character is a random human sample, scaled by per-character metrics (x-height, ascenders, descenders), sheared by a per-word slant, and timed with its recorded kinematics plus inter-character (60–170 ms) and inter-word (120–260 ms) pauses.

Geometric primitives have no human source, so we synthesise their kinematics. Each stroke is a chain of sub-movements joined at corners with a short dwell (25–70 ms). A sub-movement of path length Λ follows the minimum-jerk profile [7]

$$s(\tau) = \Lambda\,(10\tau^3 - 15\tau^4 + 6\tau^5), \quad \tau = t/T, \qquad T = \frac{90 + 11\sqrt{\Lambda}}{v}\,u \text{ ms}, \; u \sim \mathcal{U}(0.85, 1.15), \quad (9)$$

with a primitive-specific speed factor v. A smooth perpendicular wobble (three random sinusoids, tapered to zero at the end points) imitates an unsteady hand, and closing corners of boxes overshoot slightly.

3.5 Text-to-drawing pipeline

The motor model consumes an image; to start from a text request we prepend two stages (Algorithm 1).

Stage 1: planner. A frozen Qwen2.5-3B-Instruct, prompted with a system message containing three caption examples in the style of the scene grammar (Appendix B), maps a free-form request to JSON with a caption and up to eight short label strings. Decoding is greedy.

Algorithm 1 Text-to-drawing inference.

Require: request r

$(caption, [s_1, \ldots, s_n]) \leftarrow \textsc{Planner}(r)$ ▷ Qwen2.5-3B-Instruct, few-shot

$I \leftarrow \textsc{Sana+LoRA}(\textsc{Style}(caption))$ ▷ 512 px, 20 steps

$[b_1, \ldots, b_m] \leftarrow \textsc{DetectBanners}(I)$ ▷ reading order

$\mathcal{A} \leftarrow \{(a, s_a, b_a)\}_{a<\min(n,m)}$

$I' \leftarrow \textsc{Canonicalise}(I, [b_1, \ldots, b_m])$ ▷ skeleton, training width

$e_{1:L} \leftarrow \textsc{Stylus.generate}(I', \textsc{Prompt}(\mathcal{A}))$

return $e_{1:L}$ ▷ rendered as an animation

Stage 2: sketcher. SANA-600M at 512 px [29], fine-tuned with a LoRA [13] on clean renders of synthetic scenes in which *every* anchor is a grey banner, conditioned on the scene caption with a fixed style prefix and suffix. Diffusion models do not spell reliably; the banners reserve space for text that Stage 3 writes. Training uses the flow-matching objective [18, 20] with linear interpolation $\mathbf{z}_\sigma = (1-\sigma)\mathbf{z}_0 + \sigma\boldsymbol{\epsilon}$, velocity target $\boldsymbol{\epsilon} - \mathbf{z}_0$ and logit-normal σ [6].

Banner detection. Grey pixels in a band around the banner grey level are cleaned by a morphological opening. A connected component is accepted as a banner if it is large enough, wider than tall, at least 70% filled, does not touch the image border, and has a flat fill (standard deviation of its grey values below 9), which rejects shading and hatching. Banners are sorted into reading order and assigned to the planned labels in order. On 40 synthetic test scenes rendered at the sketcher resolution, this recovers 136 of 138 banners.

Sketch canonicalisation. Diffusion sketches contain paper texture, shading and thick, uneven lines that never occur in the renders the motor model was trained on. Before Stage 3 we therefore binarise the sketch with an adaptive threshold, remove the banner areas and small specks, reduce the ink to its one-pixel skeleton, and redraw it at the training stroke width on a white canvas; detected banners are repainted as flat grey bars. This maps Stage 2 outputs onto the clean-render distribution at the cost of the stroke width and of small details.

Stage 3: motor. Stylus receives the sketch (resized to 384 px) and the anchor prompt and generates the event sequence, writing each label as a `TEXT_START`(a) … `TEXT_END` span.

4 Experimental Setup

Data. Table 4 summarises the corpus produced by the data pipeline of §3.4. Training samples are drawn from a weighted mixture of sources: synthetic scenes 45%, Quick, Draw! 33%, MathWriting 18%, Calliar 4%. Test sets are the hash-defined 1% splits of Quick, Draw! and Calliar, the 2000 smoothest expressions of the MathWriting test split, and 1000 scenes generated from disjoint seeds.

Training. All parameters, including the vision encoder and the backbone, are trained with AdamW and per-group learning-rate multipliers (Table 5). Samples are bucketed by length into micro-batches under a budget of padded tokens (vision tokens, prompt tokens and events). Training runs under a hard wall-clock budget; the cosine schedule is driven by the larger of step and wall-clock progress, so it always anneals by the end of the budget.

The motor model was trained in two phases on one GPU. *Phase 1* (60 min) used the architecture without the pointer prior and the original renderer (stroke width 1.2–3.6 px). Guided by close visual checks of generated drawings every 30 min, we observed that the model learned the sequence prior (stroke order, text placement, termination) but placed strokes only coarsely, and

Table 4: Training corpus after filtering. Events exclude `BOS`/`EOS`. Numbers are taken from `data/processed/stats.json`.

Source	Filter	Train	Test	Events	Weight
Quick, Draw! [14]	recognised, 2–30 strokes, 0.3–25 s, smoothest 45%	600,590	6,111	21.6 M	0.33
MathWriting [8]	≤60 strokes, smoothest 30% (≤60k)	60,000	2,000	2.9 M	0.18
Calliar [1]	≤80 strokes	2,241	249	0.15 M	0.04
Synthetic scenes	procedural grammar (Sec. 3.4)	150,000	1,000	32.1 M	0.45
Icon bank	60 smoothest per category	20,469 icons, 345 categories			
Glyph bank	shape-consistent, smoothest 60%, ≤150 per char.	2,339 glyphs, 80 characters			

Table 5: Training configuration. Motor-model values are those of phase 2; phase 1 used a 40k-token budget and 300 warm-up steps, and its cosine schedule (6000 steps or 95 min) was cut at 61 min, at a learning rate of 1.5×10^{-4}. Phase 2 ran as three resumed runs (§4).

	Stage 3: motor model	Stage 2: sketch LoRA
Initialisation	SigLIP So400m/14@384, Qwen2.5-0.5B	SANA-600M 512 px
Trainable	all parameters	LoRA, rank 32, $\alpha = 32$
Objective	Eq. (7)	flow matching, logit-normal σ
Optimiser	AdamW, $\beta = (0.9, 0.95)$	AdamW
Peak learning rate	2×10^{-4} (new layers)	1×10^{-4}
LR multipliers	vision 0.25, backbone 0.5	–
Weight decay	0.05 (pretrained), 0 (new)	10^{-4}
Schedule	100 warm-up, cosine to 10%	100 warm-up, cosine
Schedule length	wall-clock budget per run	1500 steps
Batch	≤20k padded tokens, ≤64 samples	32 images
Gradient clipping	1.0	1.0
Precision	bf16 autocast	bf16 autocast
Training data	mixture of Table 4	12,000 scene renders
Training time	61 + 31 + 91 + 52 min	7.6 min
Steps reached	4100 + 2400 + 6930 + 4080	1500
Samples / events seen	377k / 44.9M	48k images

that its end-point loss on Quick, Draw! had nearly stopped improving. For *phase 2* we added the pointer prior (§3.2), whose weights start near-uniform, thinned the rendered strokes, and resumed from the phase-1 checkpoint. Phase 2 ran as three consecutive resumed runs: 30 min, then 91 min at peak learning rate, and a final 52 min run from the last checkpoint with a fresh warm-up and cosine decay to 10% of the peak rate. Each resumed run restarts the optimiser state. In total the motor model trained for about 3.9 h on one GPU. All results below are for the final phase-2 model unless noted.

Hardware and cost. All GPU work ran on a single rented NVIDIA H100 NVL (94 GB) in one session of 5.4 h, which includes uploading the data, the four motor-model runs (233 min of training), the sketch LoRA (7.6 min), the visual checks during training, the evaluation and the demos. Data preprocessing (filtering, Bézier fitting, scene generation) runs on CPU and needs no GPU. Latency figures are given in §5.

4.1 Baselines

Skeleton tracing. A non-learned baseline that reflects the standard way of animating a bitmap. The image is binarised (threshold 160), thinned with the Zhang–Suen algorithm [31],

and the skeleton graph is cut into paths at junctions and end points; paths shorter than 4 px are dropped. Paths are ordered greedily (nearest unvisited path end, starting at the top-left corner) and drawn at a constant speed of 0.8 canvas units per ms with 150 ms pen-up gaps. This baseline isolates what the learned model adds: stroke segmentation, order and kinematics. It cannot write text into banners.

Controlled comparisons. Separate training runs per design choice were beyond our compute budget. We report two comparisons that the training history and the decoder allow: (i) *pointer prior*: held-out losses and generations of the phase-1 model against phase 2 at matched training time (this comparison also includes the renderer change and is therefore not a clean ablation); (ii) *sampling temperature* 0.3 vs. 0.6 for the same checkpoint.

4.2 Metrics

Predicted and reference trajectories are rendered at 384 px with width 2 px; ink pixels are those darker than 128. Distances are in canvas units (1/1000 of the image width).

- **Chamfer distance** ↓: the mean of the two directed mean distances between ink pixels of prediction and reference, computed with distance transforms.
- **Ink F1@5 / F1@10** ↑: F1 of ink-pixel precision and recall with a matching tolerance of 5 or 10 canvas units.
- **Order Kendall** τ ↑ [15]: predicted and reference strokes are matched by Hungarian assignment [16] on centroid distance, and τ is the rank correlation between the drawing positions of matched pairs.
- **Log dimensionless jerk** (Eq. 8), reported next to the value of the human reference; closer to human is better.
- **Duration ratio**: total predicted drawing time over human drawing time (ideal 1).
- **Stroke ratio**: number of predicted strokes over human strokes (ideal 1).

For text anchors we report the **anchor recall**: the fraction of a sample's anchors for which the model opens a text span (`TEXT_START` with that id). It measures whether every label is written, not whether it is legible; legibility is judged visually (§5). For the full pipeline we report banner-detection recall on synthetic renders and show qualitative results; we did not run a user study.

5 Results

Reconstruction of held-out drawings. Table 6 compares Stylus with the skeleton-tracing baseline on the four test splits. Inputs are clean renders of the human drawing (no degradation, labels visible); for scenes and MathWriting the anchor strings are given in the prompt. We draw one sample per input at temperature 0.6 with top-$k = 20$, using 40 scenes and 100, 100 and 60 inputs from Quick, Draw!, MathWriting and Calliar.

The baseline traces the very image it is scored against, so its geometric scores are near-perfect by construction; Stylus has to re-generate every stroke and is far less precise (Chamfer 7.9 on scenes, 8.6 on MathWriting, 23.0 on Quick, Draw!, 30.6 on Calliar). The comparison that the model is built for is the *process*. Here Stylus orders strokes more like the human on every test set: Kendall τ is 0.90 vs. 0.69 on MathWriting, 0.75 vs. 0.26 on scenes, 0.19 vs. 0.10 on Quick, Draw! and 0.29 vs. −0.11 on Calliar. Its log dimensionless jerk is closer to the human value on MathWriting (4.44 vs. 4.55, human 4.15) and Quick, Draw! (8.75 vs. 7.03, human 9.36),

Table 6: Reconstruction of held-out drawings from clean renders. One sample per input at temperature 0.6; mean over 40–100 test inputs per source. ↓/↑: lower/higher is better. Log dimensionless jerk (DLJ) is better when closer to the reference row; duration and stroke ratios are ideally 1. Bold marks the better of the two methods (without the ink budget). "+ ink budget": the same model and inputs decoded with the optional ink budget ($c = 24$, §3.2). Anchor recall: fraction of labels for which the model opens a text span. Latency is not shown here (see text).

Test set	Method	Chamfer↓	F1@5↑	F1@10↑	τ↑	log DLJ	Dur. ratio	Str. ratio	Anchors↑
Quick, Draw!	Skeleton tracing	**0.3**	**1.00**	**1.00**	0.10	7.03	**1.04**	**2.86**	–
	Stylus-0.9B	23.0	0.42	0.56	**0.19**	**8.75**	1.83	3.45	–
	+ ink budget	22.6	0.42	0.56	0.20	8.68	1.73	2.54	–
	Human	–	–	–	–	9.36	1	1	–
MathWriting	Skeleton tracing	**0.4**	**0.99**	**1.00**	0.69	4.55	0.73	1.38	–
	Stylus-0.9B	8.6	0.54	0.72	**0.90**	**4.44**	**0.92**	**1.02**	1.00
	+ ink budget	8.6	0.54	0.72	0.90	4.44	0.92	1.02	1.00
	Human	–	–	–	–	4.15	1	1	–
Calliar	Skeleton tracing	**0.5**	**0.99**	**0.99**	-0.11	5.56	**2.52**	**3.16**	–
	Stylus-0.9B	30.6	0.35	0.46	**0.29**	**9.11**	3.75	3.27	–
	+ ink budget	28.2	0.36	0.47	0.31	9.10	2.92	2.24	–
	Reference	–	–	–	–	9.47	1	1	–
Scenes	Skeleton tracing†	**0.5**	**0.98**	**0.99**	0.26	**6.18**	0.57	**1.18**	–
	Stylus-0.9B	7.9	0.70	0.82	**0.75**	6.22	**1.33**	2.00	0.96
	+ ink budget	5.7	0.71	0.83	0.79	6.23	1.17	1.26	1.00
	Reference	–	–	–	–	5.80	1	1	–

†Scene inputs are rendered with their labels visible, so the baseline traces the label ink but cannot know which label it is writing. Calliar timing is synthetic (constant speed), so its kinematic columns do not compare against human motion. The baseline traces the very image it is scored against, which makes its geometric scores near-perfect by construction.

and on par with the baseline on scenes (6.22 vs. 6.18, reference 5.80). On MathWriting, total drawing time and stroke count are close to human (0.92 and 1.02 of the reference), and every label is written (1.00 anchor recall); on scenes, 96% of labels are written. On Quick, Draw! and Calliar, Stylus uses more strokes and takes longer than the human.

With the optional ink budget (§3.2), scene reconstructions improve clearly on the same inputs (Chamfer 7.9 → 5.7, stroke ratio 2.00 → 1.26, all samples terminate and all labels are written), while MathWriting is unaffected because the budget never binds there.

Latency. Generation runs at one transformer step per event. With eight evaluation processes sharing one H100 NVL, the median time per drawing was 2.8 s for MathWriting (48 events on average), 2.7 s for Quick, Draw! (85), 3.9 s for Calliar (216) and 15.3 s for scenes (413); these are upper bounds for a dedicated GPU.

Pointer prior. Table 7 shows the held-out end-point losses around the change from phase 1 to phase 2. In its last 500 steps, phase 1 improved the Quick, Draw! end-point loss only from 5.57/5.50 to 5.56/5.47 nats (x_3/y_3), and visual checks showed drawings that ignored the input. Within the first 500 steps with the pointer prior, the same losses fell to 5.25/4.53; the largest drop is in y_3, which the pointer conditions on the patch column of x_3. Generated drawings started to follow the input from the first checkpoint on. The phase change also thinned the rendered strokes, so part of the improvement may come from more legible inputs; we did not have the budget to separate the two effects.

Figure 3 shows training and held-out losses over all runs. Held-out losses keep falling for Quick, Draw!, MathWriting and scenes until the end. Calliar, the smallest source (4% of sam-

Table 7: Held-out end-point losses (nats per event, x_3 / y_3) before and after adding the pointer prior. Phase 1 has no pointer; phase 2 resumes from the 59-min checkpoint with the pointer prior and thinner renders (so the comparison is not a clean ablation). Times are training minutes on one GPU since the start of phase 1. Values from `train_log.jsonl` of each run.

	Quick, Draw!	MathWriting	Scenes	Calliar
Phase 1, 52 min	5.57 / 5.50	4.72 / 4.85	3.36 / 3.29	5.02 / 5.21
Phase 1, 59 min	5.56 / 5.47	4.69 / 4.79	3.28 / 3.20	5.00 / 5.19
+ pointer, 66 min	5.25 / 4.53	4.57 / 4.45	3.22 / 3.09	4.97 / 4.67
+ pointer, 85 min	4.98 / 4.33	4.37 / 4.29	3.05 / 2.92	4.74 / 4.42
Final, 233 min	4.58 / 4.06	3.99 / 3.99	2.74 / 2.66	4.98 / 4.56

ples), is the exception: its held-out loss rises again during the final annealed run (28.7 → 30.7 nats), which matches its poor reconstructions.

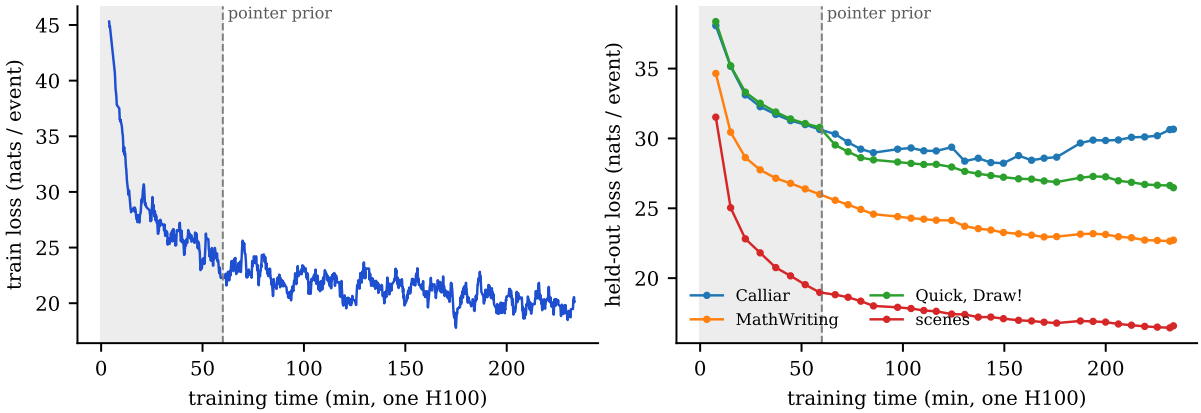


Figure 3: Training loss (left, smoothed) and held-out loss per test set (right) over cumulative training time on one GPU. Grey: phase 1 without the pointer prior. Phase 2 consists of three resumed runs; the small bump near 183 min is the warm restart of the final annealed run.

Sampling temperature. Table 8 compares temperatures 0.6 and 0.3 for the final checkpoint. Lower temperature does not make the model more precise; it makes degenerate loops more likely. At 0.3, only half of the scene samples terminate within the event budget and Quick, Draw! drawings use 27× as many strokes as the human; MathWriting is nearly unchanged. We use 0.6 throughout.

Kinematics. Figure 4 shows the pen speed over one MathWriting drawing and the distribution of segment durations over all saved samples. The baseline moves at constant speed. Stylus produces varying speeds and pauses between strokes, and above ≈ 30 ms its segment durations follow the human distribution closely. Below that, it has a second mode at 10–25 ms that the human data does not have: the model splits strokes into many short, fast segments, the fragmentation that also shows in the stroke ratios.

Qualitative results. Figure 5 shows test inputs, the two methods and the human reference, two inputs per source, the first saved samples in evaluation order (not selected). Formulas are re-drawn symbol by symbol in the human order, with recognisable but rough glyphs. Scenes are reproduced as a structure, with boxes, arrows and labels in a plausible order, but dense icons are dropped or replaced, and some primitives (here, the circles around nodes) are missing.

Table 8: Effect of the sampling temperature for the final checkpoint (half the test inputs of Table 6 at the alternative temperature). EOS: fraction of samples that terminate within the event budget.

Test set	T	Chamfer↓	τ↑	Str. ratio	Dur. ratio	EOS↑
Quick, Draw!	0.3	16.7	0.18	26.94	9.25	0.84
	0.6	23.0	0.19	3.45	1.83	0.98
MathWriting	0.3	9.1	0.90	1.04	0.91	1.00
	0.6	8.6	0.90	1.02	0.92	1.00
Calliar	0.3	25.5	0.21	9.93	9.08	0.73
	0.6	30.6	0.29	3.27	3.75	0.97
Scenes	0.3	23.7	0.63	4.87	2.96	0.50
	0.6	7.9	0.75	2.00	1.33	0.93

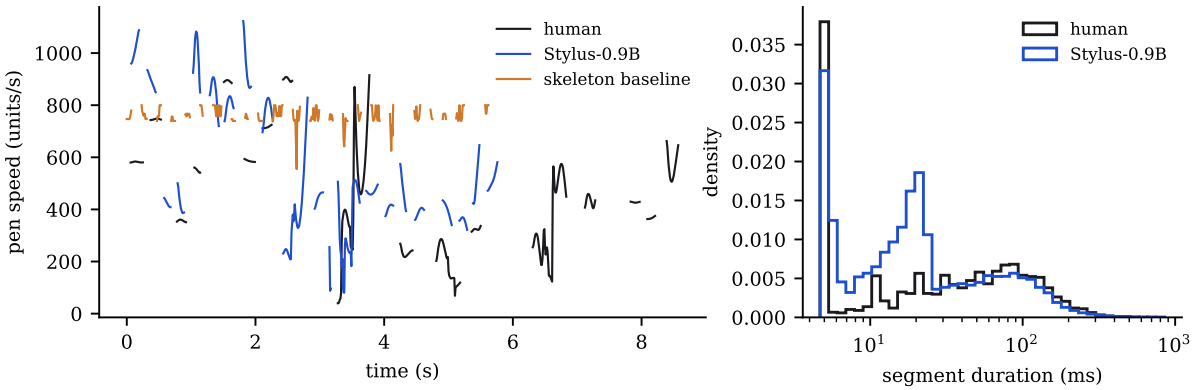


Figure 4: Kinematics. Left: pen speed (canvas units per second) over time for one MathWriting test drawing, for the human, Stylus and the baseline; gaps are pen-up periods. Right: distribution of segment durations of Stylus and the human references over all saved test samples, one bar per duration bin.

Quick, Draw! doodles keep their outline but lose details. Calliar fails: the dense, overlapping calligraphy is replaced by a few unrelated strokes.

Full pipeline. Figure 6 shows end-to-end results for text requests. Figure 7 compares SANA-600M with and without the whiteboard LoRA for the same prompts and seeds. The LoRA turns hatched pseudo-diagrams with illegible pseudo-text into clean boxes, arrows and cylinders with grey label banners; on these, banner detection found all 9 banners of the two training samples we inspected. Simple objects (cat, rocket) are preserved, but a detailed object (a bicycle) degrades, since the LoRA only saw diagrams. End to end, diagram requests work best: the web-app and ML-pipeline requests produce the planned boxes in the planned order with partly legible labels. Requests for objects fail: the planner turns them into labelled diagrams, and the motor model collapses into scribbles on the resulting icons. For the banana load balancer, the boxes, arrows and the first labels are drawn correctly, but the banana icons on the arrows trigger dense scribble clusters, which the ink budget reduces but does not remove. The full pipeline takes a median of 8.6 s (planner), 11.5 s (sketcher, 20 steps) and 26.6 s (motor, median 782 events) per request on a shared H100.

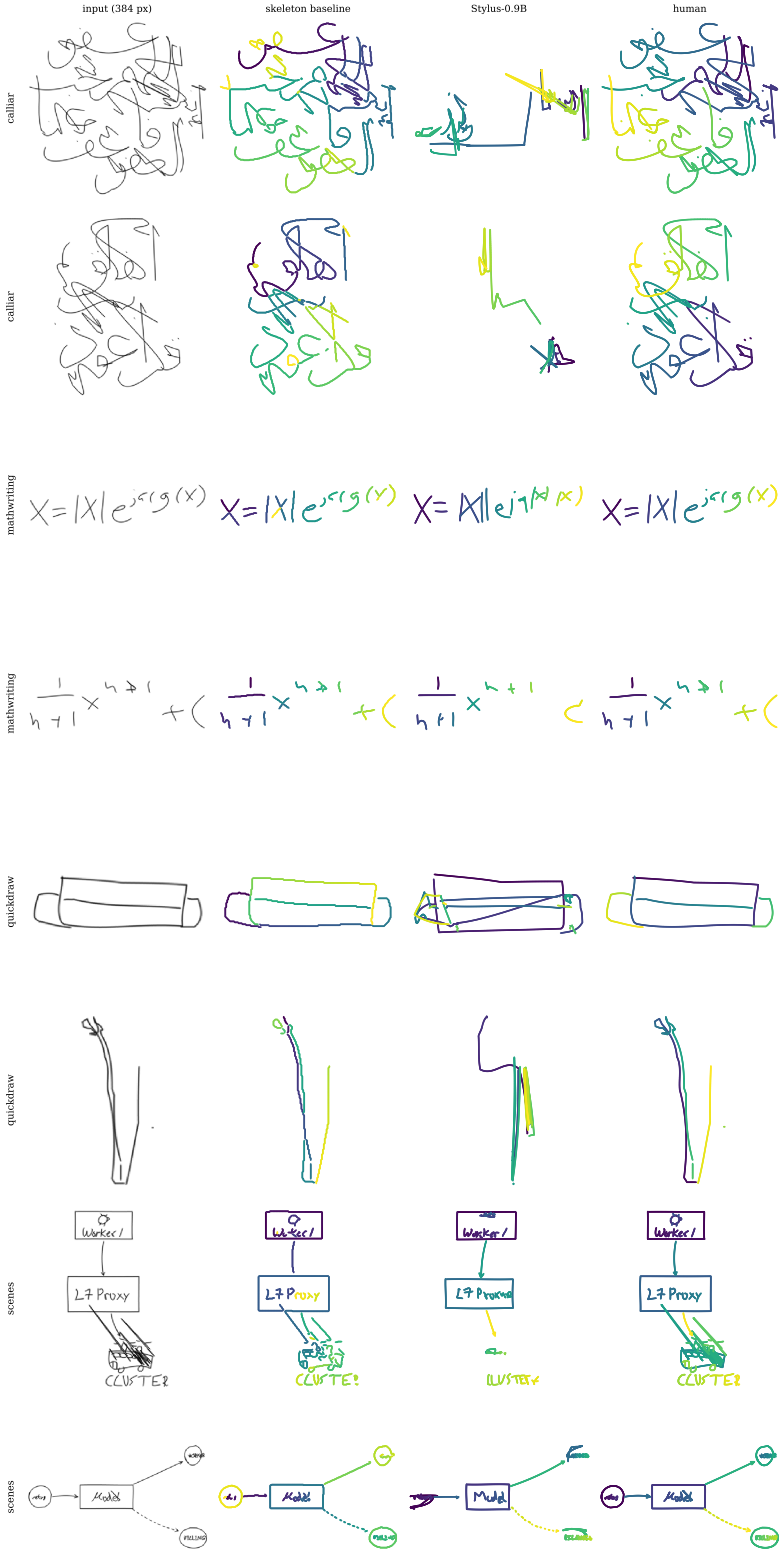


Figure 5: Qualitative comparison on held-out test inputs (first two saved samples per source, not selected). Columns: input render, skeleton baseline, Stylus ($T = 0.6$), human reference. Stroke colour encodes drawing time from early (dark blue) to late (yellow). Each row is cropped to the human drawing.

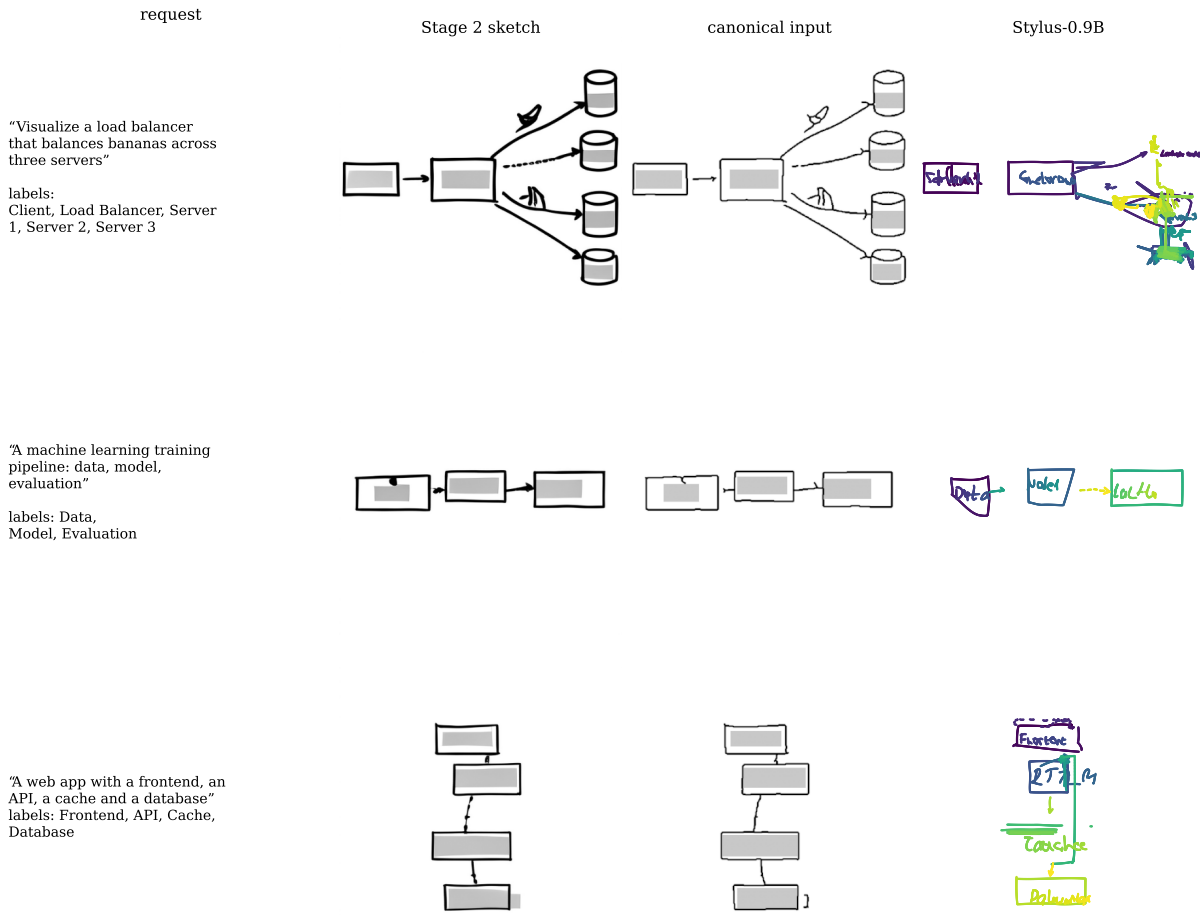


Figure 6: Text-to-drawing pipeline (seed 1, ink budget on, no selection). From left: request and the planner’s labels, Stage 2 sketch (SANA-600M + LoRA), canonicalised motor input, and the Stylus drawing (colour = time).

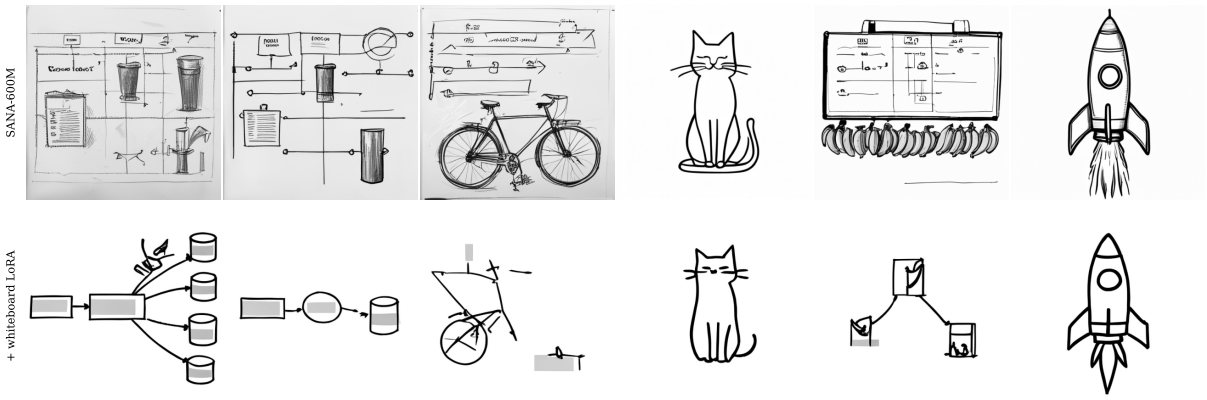


Figure 7: Stage 2 before (top) and after (bottom) the whiteboard LoRA, same prompt and seed per column. Columns 1–3 use planner-style captions with the sketch style suffix (banana load balancer with three servers; input–model–output flow; annotated bicycle); columns 4–6 are free prompts (cat; banana load balancer; rocket).

6 Limitations and Broader Impact

Observed failure modes. We inspected generated drawings visually throughout training, and several weaknesses remain clear in the final model. (i) *Legibility:* labels are written at the right place and in a plausible order, but letter shapes are often rough and words are frequently not legible. (ii) *Fragmentation:* strokes are split into more and shorter segments than in the human reference, which inflates stroke counts and total drawing time (Table 6). (iii) *Free-form doodles:* Quick, Draw! drawings are reproduced much more coarsely than diagrams and formulas; long, irregular contours are shortened or replaced by plausible but different shapes, and some samples collapse into dense scribbles at one location. (iv) *Sample variance:* quality varies strongly between samples and between nearby checkpoints, so small evaluation sets give noisy estimates; our guided checks with 8 inputs per set showed visibly worse drawings at one checkpoint although the held-out loss kept improving. (v) *Sketch canonicalisation:* for diffusion sketches, the skeleton-based canonicalisation (§3.5) removes shading and paper texture but also destroys small handwriting, so text must come from banners, not from the sketch.

Synthetic geometry. Boxes, cylinders, circles, diamonds and arrows in the training scenes are not human ink. Their kinematics follow a minimum-jerk model with hand-set constants (Eq. 9), a perpendicular wobble and fixed dwell ranges, and their drawing order follows hand-written rules. The model can at best reproduce these rules for diagram structure; whether they match how people draw diagrams has not been measured against recorded whiteboard sessions, which we do not have.

Composed handwriting. Labels in scenes are assembled from single-character glyphs cut out of MathWriting inks by many different writers. Within one word, characters can therefore differ in style and size, and ligatures between characters are absent; the model learns this inconsistent mixture. The glyph bank is limited to characters present in the symbol annotations, and other characters are dropped from labels. Calliar has no timestamps, so its timing is synthetic.

Representation. The event format has no pen pressure, tilt or stroke width; all strokes are rendered with uniform width. Durations are clipped to 5–800 ms, so longer pauses are not represented, and timing inside a Bézier segment is linear in the curve parameter rather than recorded velocity. Coordinates are quantised to 1/1000 of the canvas. At most 16 text anchors and 1500 generated events are supported.

Pipeline dependence. Stage 3 can only redraw what Stage 2 draws. If the diffusion model omits, merges or misplaces banners, writes illegible pseudo-text, or produces structures outside the training distribution of the degradation operator, banner detection and the motor model inherit these errors. The planner is used few-shot without fine-tuning and may produce captions that the sketcher does not follow. The sketch LoRA is trained only on synthetic scene renders, which limits the range of diagram styles.

Evaluation. Our metrics compare against a single human reference per drawing, although many stroke orders are plausible; Kendall τ after centroid matching is a coarse proxy for order quality. Synthetic test scenes share the generator with the training scenes. We did not run a user study, so whether people perceive the drawings as human-made is not measured.

Compute and scope. This is a proof of concept. Each stage is trained once, within a budget of about 5.4 GPU-hours on a single rented GPU, without hyperparameter search or multiple seeds, so reported differences may be within run-to-run variance. The motor model

saw about 377k samples, less than one pass over Quick, Draw! alone, and its held-out losses were still falling at the end of training (Figure 3). We have not measured how quality scales with compute; our estimate is that precise geometry and legible handwriting require one to two orders of magnitude more training, i.e. weeks on high-end multi-GPU hardware, together with a higher input resolution for text and feedback of the model's own canvas.

Broader impact. A model that produces human-like pen trajectories could be used to make machine-generated drawings or handwriting appear hand-made, for example to imitate a person's handwriting on a digital signature pad or to pass off generated content as human work in contexts that rely on online handwriting as evidence of authorship. Stylus does not model the style of a particular writer and was not trained on signatures, which limits this risk but does not remove it. Intended uses are educational animations, explanatory whiteboard videos and accessible diagram presentation. The training data are public datasets released for research: Quick, Draw! (CC BY 4.0), MathWriting (CC BY-NC-SA 4.0) and Calliar (MIT); the pretrained components SigLIP (Apache-2.0), Qwen2.5-0.5B and Qwen2.5-3B-Instruct (Apache-2.0 and the Qwen research licence, respectively) and SANA-600M keep their own terms. Because MathWriting is non-commercial, we release the Stylus weights for non-commercial research use only; the code is released under Apache-2.0.

7 Conclusion

We presented Stylus-0.9B, a motor model that converts a raster sketch and a list of text anchors into a timed sequence of pen events. Its training signal requires no paired data: trajectories from human ink and from a procedural scene grammar are rendered through a stochastic degradation operator and the model learns to invert it. An event-level sequence with an autoregressive field decoder keeps sequences short while modelling dependencies inside an event. Combined with an LLM planner and a diffusion sketcher that leaves room for text, the model turns a text request into a hand-drawn diagram animation. On held-out data it orders strokes much more like humans than tracing the image, and its motion is closer to human on formulas and doodles, but its geometry is coarse, its strokes are fragmented and its handwriting is often illegible; the change that mattered most was a pointer prior that ties end points to image patches. As a proof of concept, the model shows that this kind of motor model can be trained cheaply from public data; finishing it is a matter of weeks of training on high-end hardware. Directions for future work include recording real whiteboard sessions to replace the rules for diagram order and primitive kinematics, writer-consistent handwriting, pressure and width modelling, and training the sketcher and motor model jointly.

References

[1] Zaid Alyafeai, Maged S. Al-shaibani, Mustafa Ghaleb, and Yousif Ahmed Al-Wajih. Calliar: An online handwritten dataset for Arabic calligraphy, 2021. arXiv:2106.10745.

[2] Taylor Archibald, Mason Poggemann, Aaron Chan, and Tony Martinez. TRACE: A differentiable approach to line-level stroke recovery for offline handwritten text. In *International Conference on Document Analysis and Recognition (ICDAR)*, 2021.

[3] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. DeepSVG: A hierarchical generative network for vector graphics animation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[4] Vage Egiazarian, Oleg Voynov, Alexey Artemov, Denis Volkhonskiy, Aleksandr Safin, Maria Taktasheva, Denis Zorin, and Evgeny Burnaev. Deep vectorization of technical drawings, 2020. arXiv:2003.05471.

[5] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B. Tenenbaum. Learning to infer graphics programs from hand-drawn images, 2017. arXiv:1707.09627.

[6] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, pages 12606–12633, 2024.

[7] Tamar Flash and Neville Hogan. The coordination of arm movements: An experimentally confirmed mathematical model. *Journal of Neuroscience*, 5(7):1688–1703, 1985. doi: 10.1523/JNEUROSCI.05-07-01688.1985.

[8] Philippe Gervais, Anastasiia Fadeeva, and Andrii Maksai. MathWriting: A dataset for handwritten mathematical expression recognition, 2024. arXiv:2404.10690.

[9] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, pages 15706–15734, 2024.

[10] Alex Graves. Generating sequences with recurrent neural networks, 2013. arXiv:1308.0850.

[11] David Ha and Douglas Eck. A neural representation of sketch drawings. In *International Conference on Learning Representations (ICLR)*, 2018.

[12] Neville Hogan and Dagmar Sternad. Sensitivity of smoothness measures to movement duration, amplitude, and arrests. *Journal of Motor Behavior*, 41(6):529–534, 2009.

[13] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

[14] Jonas Jongejan, Henry Rowley, Takashi Kawashima, Jongmin Kim, and Nick Fox-Gieg. The Quick, Draw! A.I. experiment and dataset. https://quickdraw.withgoogle.com; data: https://github.com/googlecreativelab/quickdraw-dataset, 2016.

[15] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. doi: 10.1093/biomet/30.1-2.81.

[16] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955. doi: 10.1002/nav.3800020109.

[17] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics*, 39 (6), 2020. doi: 10.1145/3414685.3417871.

[18] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

[19] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[20] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations (ICLR)*, 2023.

[21] Réjean Plamondon. A kinematic theory of rapid human movements. Part I: Movement representation and generation. *Biological Cybernetics*, 72:295–307, 1995.

[22] Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen2.5 technical report, 2024. arXiv:2412.15115.

[23] Leo Sampaio Ferraz Ribeiro, Tu Bui, John Collomosse, and Moacir Ponti. Sketchformer: Transformer-based representation for sketched structure. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[24] Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. StarVector: Generating scalable vector graphics code from images and text. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16175–16186, 2025.

[25] Philip J. Schneider. An algorithm for automatically fitting digitized curves. In Andrew S. Glassner, editor, *Graphics Gems*, pages 612–626. Academic Press, 1990.

[26] Yael Vinker, Ehsan Pajouheshgar, Jessica Y. Bo, Roman Christian Bachmann, Amit Haim Bermano, Daniel Cohen-Or, Amir Zamir, and Ariel Shamir. CLIPasso: Semantically-aware object sketching. *ACM Transactions on Graphics*, 41(4):86:1–86:11, 2022. doi: 10.1145/3528223.3530068.

[27] Qiang Wang, Haoge Deng, Yonggang Qi, Da Li, and Yi-Zhe Song. SketchKnitter: Vectorized sketch generation with diffusion models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=4eJ43EN2g6l.

[28] Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. IconShop: Text-guided vector icon synthesis with autoregressive transformers. *ACM Transactions on Graphics*, 42(6), 2023.

[29] Enze Xie, Junsong Chen, Junyu Chen, Han Cai, Haotian Tang, Yujun Lin, Zhekai Zhang, Muyang Li, Ligeng Zhu, Yao Lu, and Song Han. SANA: Efficient high-resolution image synthesis with linear diffusion transformers. In *International Conference on Learning Representations (ICLR)*, 2025.

[30] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[31] T. Y. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984. doi: 10.1145/357994.358023.

A Hyperparameters

Table 9 lists the constants of the data pipeline and the inference pipeline that are not given in the main text. All values are the defaults in the released code.

Table 9: Data and inference constants.

Component	Parameter	Value
Bézier encoder	corner turning angle / RDP tolerance	55° / 2.5 units
	fit tolerance / Newton steps / max. depth	2.0 units / 3 / 12
	first pen-up time / min. duration	60 ms / 5 ms
Samples	max. events per sample	1200
	placement box side (QD / MW / Calliar)	350–900 / 250–900 / 300–900
	test split	1% by MD5 of sample key
Jerk score	resampling rate / min. stroke duration	100 Hz / 50 ms
Calliar timing	speed / pen-up gap	0.6 units/ms (source frame) / 120 ms
Scenes	train / test scenes	150,000 / 1000
	layout probabilities (pipeline, fan-out, tree, annotated)	$0.35, 0.30, 0.20, 0.15$
	node kinds (default weights)	box 3, cylinder 1, circle 1, diamond 1, icon 2
	dashed arrow probability	0.2
	label cap height	12–34 units
	glyph slant	$\mathcal{N}(0, 0.12)$
Kinematics	sub-movement duration	$(90 + 11\sqrt{\Lambda})/v \cdot \mathcal{U}(0.85, 1.15)$ ms
	corner dwell	25–70 ms
	sampling rate	100 Hz
Motor decoding	temperature / top-k / max. events	0.6 (pipeline), 0.7 (default) / 20 / 1500
Planner	model / decoding / max. new tokens	Qwen2.5-3B-Instruct / greedy / 300
Sketcher	resolution / steps / guidance	512 px / 20 / 4.5
	LoRA target modules	`to_q, to_k, to_v, to_out.0, conv_inverted, conv_point`
	stroke width of training renders	$\mathcal{U}(2.2, 3.2)$ px
Banner detection	grey band	$[165, 230)$
	min. / max. area, min. aspect	0.15% / 6% of image, 1.2
	min. fill ratio / max. grey std. / border margin	0.7 / 9 / 1% of width
Baseline	binarisation / min. path / speed / pen-up	160 / 4 px / 0.8 units/ms / 150 ms

B Prompt templates

Motor prompt. The text given to Stylus after the image tokens. Without anchors only the first line is used. Boxes are $x_0\ y_0\ x_1\ y_1$ on the 1000 canvas. An example with two anchors:

```
Redraw this sketch stroke by stroke, like a person at a whiteboard.
Text to write:
[0] 112 431 268 470: Client
[1] 402 420 590 481: Load Balancer
```

Sketcher caption. The scene caption is wrapped as

```
whiteboard sketch, black marker on white, <caption> Text areas are blank grey banners.
```

Example captions produced by the scene grammar:

```
A hand-drawn whiteboard flow diagram, left to right: a box labeled 'Input',
then a circle labeled 'Model', then a cylinder labeled 'Output', connected by
curved arrows.
```

```
A hand-drawn whiteboard tree diagram, top to bottom: a box labeled 'Root';
a box labeled 'Left'; a box labeled 'Right', linked by arrows.

A hand-drawn whiteboard sketch: a large sketch of a cat with handwritten
annotation labels and pointer arrows.
```

Planner system prompt (Stage 1, verbatim; line breaks added).

```
You plan hand-drawn whiteboard diagrams. Reply with JSON only:
{"caption": <one sentence describing the drawing>, "labels": [<short label
strings, at most 8>]}
The caption must follow this style and mention every label in quotes, in
reading order:
- "A hand-drawn whiteboard flow diagram, left to right: a box labeled 'Input',
  then a circle labeled 'Model', then a cylinder labeled 'Output', connected
  by curved arrows."
- "A hand-drawn whiteboard architecture sketch: a box labeled 'Client' on the
  left feeds a box labeled 'Load Balancer' in the center, which fans out with
  3 curved arrows to a cylinder labeled 'Server 1', a cylinder labeled
  'Server 2', a cylinder labeled 'Server 3' on the right; each arrow carries a
  small sketched banana."
- "A hand-drawn whiteboard tree diagram, top to bottom: a box labeled 'Root';
  a box labeled 'Left'; a box labeled 'Right', linked by arrows."
Node kinds: box, circle, cylinder, diamond, or "a sketched <object>" for a
drawn object (banana, cat, truck, ...).
Labels are at most 3 words.
```

C Event grammar

A valid event sequence is generated by the following grammar, where ⟨item⟩ ranges over pen events and text spans. Within a text span the anchor id a must be smaller than the number of anchors in the prompt and may not have been written before. These constraints, and the restriction of fields to $\mathcal{F}(o)$, are enforced at decoding time by masking logits (§3.2); the training data satisfy them by construction (`events.validate`).

$$
\begin{aligned}
\langle \text{drawing} \rangle &::= \texttt{BOS}\ \langle \text{item} \rangle^*\ \texttt{EOS} \\
\langle \text{item} \rangle &::= \langle \text{stroke} \rangle \mid \texttt{TEXT_START}(a)\ \langle \text{stroke} \rangle^*\ \texttt{TEXT_END} \\
\langle \text{stroke} \rangle &::= \texttt{MOVE}(x_3, y_3, \delta)\ \texttt{CURVE}(x_1, y_1, x_2, y_2, x_3, y_3, \delta)^*
\end{aligned}
$$

The encoder always emits at least one `CURVE` after each `MOVE`; a single sample point (a dot) is encoded as a very short curve of 20 ms. The decoder grammar mask does not enforce this, so a generated `MOVE` may be followed directly by another `MOVE`.