VARIABLE-LENGTH NON-AUTOREGRESSIVE SPEECH-TO-SPEECH TRANSLATION, WITH AND WITHOUT A PRETRAINED DIFFUSION LANGUAGE MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Non-autoregressive generative approaches, such as masked diffusion language models and flow matching, have gained considerable traction in recent years: masked diffusion for text generation and code infilling, and flow matching for image generation and text-to-speech synthesis. Yet they remain rarely explored for speech-to-speech translation. A central obstacle is output length. Source and target utterances differ in length, and the relationship varies across language pairs, whereas in their standard form these models fill a canvas of output positions whose size must be fixed before decoding begins. In this work, we present several techniques for handling variable-length output in non-autoregressive speech-to-speech translation, adapting methods originally developed for other domains, such as code infilling and molecule generation. We study two settings: a large pretrained masked diffusion language model, LLaDA, together with variable-length extensions of it, and a much smaller model based on Branching Flows that does not rely on a large language model. All systems are trained and evaluated on the CVSS-C French–English benchmark. The Branching Flows model reaches 24.19 ASR-BLEU, slightly above the discrete diffusion-based model Dub-S2ST (23.35).

1 INTRODUCTION

Non-autoregressive generation has become a serious alternative to left-to-right decoding in both text and speech. In text, masked diffusion language models predict every masked position of a canvas at once and fill the canvas over a fixed number of steps (Austin et al., 2021; Lou et al., 2024; Sahoo et al., 2024; Shi et al., 2024); scaled to billions of parameters, they approach the quality of autoregressive language models of similar size (Nie et al., 2025). In speech, flow matching (Lipman et al., 2022) underlies the current generation of fast text-to-speech systems (Le et al., 2023; Mehta et al., 2024; Chen et al., 2024b). In both, the output is treated as one object that is refined in parallel, so that positions need not be written left to right.

Speech-to-speech translation (S2ST) is a natural target for these models, yet they have rarely been applied to it. Direct S2ST systems translate source speech into target speech without producing text as the final output, first as spectrograms (Jia et al., 2019) and more recently as discrete acoustic units that a vocoder turns into a waveform (Lee et al., 2022; Inaguma et al., 2023); most of them decode autoregressively. A central obstacle for a non-autoregressive decoder is output length: in translation the target may be longer or shorter than the source, and the ratio varies across language pairs and across utterances, but a canvas-based decoder must know how many positions to fill before it starts. Earlier non-autoregressive S2ST work takes its answers from non-autoregressive machine translation, where a predicted length or fertility fixes the canvas before decoding (Gu et al., 2018; Ghazvininejad et al., 2019; Huang et al., 2023), or where a decoding graph of fixed size is laid out and the output length follows from the path chosen through it (Fang et al., 2023). The two diffusion-based S2ST models closest to ours fix the canvas before decoding as well. DiffS2UT (Zhu et al., 2023) decodes several candidates of different lengths in parallel and keeps the one with the lowest perplexity. Dub-S2ST (Choi et al., 2025), built for dubbing, where the translation should keep the duration and speaking pace of the source, fixes the output canvas to the length of the source unit sequence and speed-adapts the training targets, changing the rate at which units repeat rather than their content. Both work well. This paper asks whether the decoder can decide the length itself.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Our first attempt, a diffusion decoder writing units on a fixed canvas whose training targets were padded to full length with the unit that the vocoder renders as silence, gave translations that were too short when decoded plainly; holding the silence unit back during decoding brought them close to the Dub-S2ST baseline, without reaching it (Appendix A). LLaDA (Nie et al., 2025), a pretrained masked diffusion language model, uses the same convention in fine-tuning, shorter responses padded with end-of-sequence (EOS) tokens that are masked and predicted like any other token, and its authors report a related problem, generations that end too early under low-confidence remasking, together with a corresponding remedy. This suggested our first system.

From here on, our systems keep the two-pass layout of UnitY (Inaguma et al., 2023): a speech encoder, a decoder that writes the translation as text, and a second decoder that writes the units of the target speech from that text. Both decoders are non-autoregressive at inference. The unit decoder is the same in every system that follows and takes its length from a simple predictor fitted on the text; the question in each system is how the text decoder writes the translation and decides its length.

The first system asks how far LLaDA goes with little training: attach a speech encoder through a projector, fine-tune for a few epochs on paired data, keep LLaDA's EOS convention for the targets, and decode on a fixed canvas. It receives one further epoch of training, as do the extensions that follow, so that all are compared after the same number of updates; decoded without any heuristic, it already surpasses the non-autoregressive baseline we replicate. Two recent extensions of masked diffusion then let the canvas change size while it is being decoded. DreamOn (Wu et al., 2026), developed for code infilling, lets the model expand or contract the output length on its own predictions; FlexMDM (Kim et al., 2025) generates by inserting mask tokens and unmasking them, so that the canvas grows from empty. We port both onto the same fine-tuned model and compare them with the fixed canvas under a common protocol. Section 4 reports where each of the three stands.

All of this rests on a large pretrained model. The second system asks whether a small decoder that we train ourselves can decide its own length and still reach the quality of the Dub-S2ST baselines we replicate. We train a Dub-S2ST-style diffusion-transformer decoder (Peebles & Xie, 2022) with Branching Flows (Billera et al., 2025), a general extension of discrete flow matching (Campbell et al., 2024; Gat et al., 2024) in which every position may split into several or be deleted during generation. The decoder starts from a single masked position and grows the translation to the length it needs, with no search over lengths at inference and no speed adaptation of the training targets. It starts from the same pretrained speech encoder as the first system, but no large language model is involved. On the same benchmark, with one decode-time rate multiplier chosen on the development set, it matches and slightly exceeds both the non-autoregressive Dub-S2ST model and the Dub-S2ST authors' autoregressive S2UT baseline.

All systems are trained and evaluated on the French–English portion of CVSS-C (Jia et al., 2022). The rest of the paper describes each system, then the results and what the decoding records show.

2 Background and related work

Length in non-autoregressive translation. Non-autoregressive translation methods differ in how they decide the output length. Gu et al. (2018) take it from the fertility of each source token. Mask-Predict predicts the target length with a special token, decodes the most likely lengths in parallel and keeps the candidate with the highest average log-probability; its decoding repeatedly re-masks the words the model is least confident about (Ghazvininejad et al., 2019). The Directed Acyclic Transformer lays out its hidden states as a graph whose size is set in advance and in which every path is a translation, so that the chosen path decides the length (Huang et al., 2022). Insertion-based and edit-based models instead change the length while decoding: the Insertion Transformer inserts tokens at several positions at once, and the Levenshtein Transformer combines insertion and deletion (Stern et al., 2019; Gu et al., 2019).

Direct speech-to-speech translation. Translatotron maps source spectrograms to target spectrograms with no text in between (Jia et al., 2019). S2UT replaces the spectrogram target with discrete units obtained from a self-supervised encoder of the target speech (Lee et al., 2022), and UnitY generates text first and units second, with one decoder for each (Inaguma et al., 2023). Non-autoregressive S2ST systems followed. TranSpeech masks and re-predicts the units for a few cycles, with the length taken from a predictor and, optionally, a beam over lengths (Huang et al., 2023).

DASpeech is a two-pass non-autoregressive system built on the Directed Acyclic Transformer: a graph-based linguistic decoder generates the target text as a phoneme sequence, and an acoustic decoder in the style of FastSpeech 2 generates the mel-spectrogram from the hidden states of the chosen path (Fang et al., 2023).

Masked diffusion and discrete flow matching. D3PM developed discrete diffusion with structured corruption processes, including absorbing states that connect diffusion to masked generation (Austin et al., 2021). Discrete flow matching models discrete generation through continuous-time Markov chains along chosen probability paths, with discrete diffusion as a special case (Campbell et al., 2024; Gat et al., 2024).

Variable-length diffusion and flow matching. A fixed-length canvas is a recognised limitation of these models. DreamOn adds two length-control states so that the model expands or contracts its output on its own predictions (Wu et al., 2026), FlexMDM generates by inserting mask tokens and unmasking them (Kim et al., 2025), and Branching Flows lets the elements of a state branch and die during generation, at rates the model learns (Billera et al., 2025); Section 3 describes each as we adapt it to speech translation.

3 Systems and experimental setup

The systems of this section all write English text from French speech and then render the text into speech. They share the architecture and released initialisation of the speech encoder, which each system fine-tunes under its own objective, and they share one trained text-to-unit model, one vocoder and one evaluation protocol; they differ in the text decoder and in how it is trained. The external baselines, the Dub-S2ST model and its authors' autoregressive baseline, and the pilot of Appendix A write units directly, without text, and bypass this text-to-unit stage. This section describes the shared parts and then each system; Appendix B gives the decoding of each system on a test sentence.

3.1 Shared front end and back end

The speech encoder is the 12-layer Conformer encoder, with 256-dimensional states, of a UnitY (Inaguma et al., 2023) French–English model trained on CVSS-C and released with StreamSpeech (Zhang et al., 2024).[1] It consumes 80-dimensional log-mel filterbank features with global cepstral mean and variance normalisation and subsamples by a factor of four. Every system initialises it from the released checkpoint and fine-tunes it, so the trained encoder weights differ from system to system. The text written by the text decoder is rendered into speech by one text-to-unit stage, trained once and used unchanged for every system of this section. A length predictor, a rounded linear rule fitted on the training set, sets the number of units $\hat{L}$ from the numbers of pieces, characters and words of the text, and a Transformer encodes the text (Appendix C). A diffusion transformer, the Dub-S2ST decoder with absolute position embeddings added to its rotary ones, generates the units on a canvas of $\hat{L}$ slots while cross-attending to the encoded text; the same decoder architecture is used by the Branching Flows system and by the pilot. The units are frame-level mHuBERT units with consecutive repetitions retained, one unit per 20 ms, so they carry their own durations; they go to the unit HiFi-GAN vocoder with its duration predictor disabled, and the audio is scored by ASR-BLEU. Appendix C compares this stage with other ways of turning the text into speech.

3.2 LLaDA: a pretrained masked diffusion language model

Model. Our starting checkpoint is the pretrained LLaDA-8B-Base (Nie et al., 2025), a masked diffusion language model: a forward process masks the tokens of a text, and a Transformer is trained to predict the masked tokens. LLaDA's supervised fine-tuning procedure handles variable response length through EOS padding: EOS tokens are appended to the shorter responses of a mini-batch, are treated as part of the response and included in the training objective, and are removed from the generated output during sampling, so the model learns to end its response by generating EOS (Nie et al., 2025, App. B.1). This is the behaviour we want from a speech translator: the decoder

[1] https://huggingface.co/ICTNLP/StreamSpeech_Models/blob/main/unity.fr-en.pt

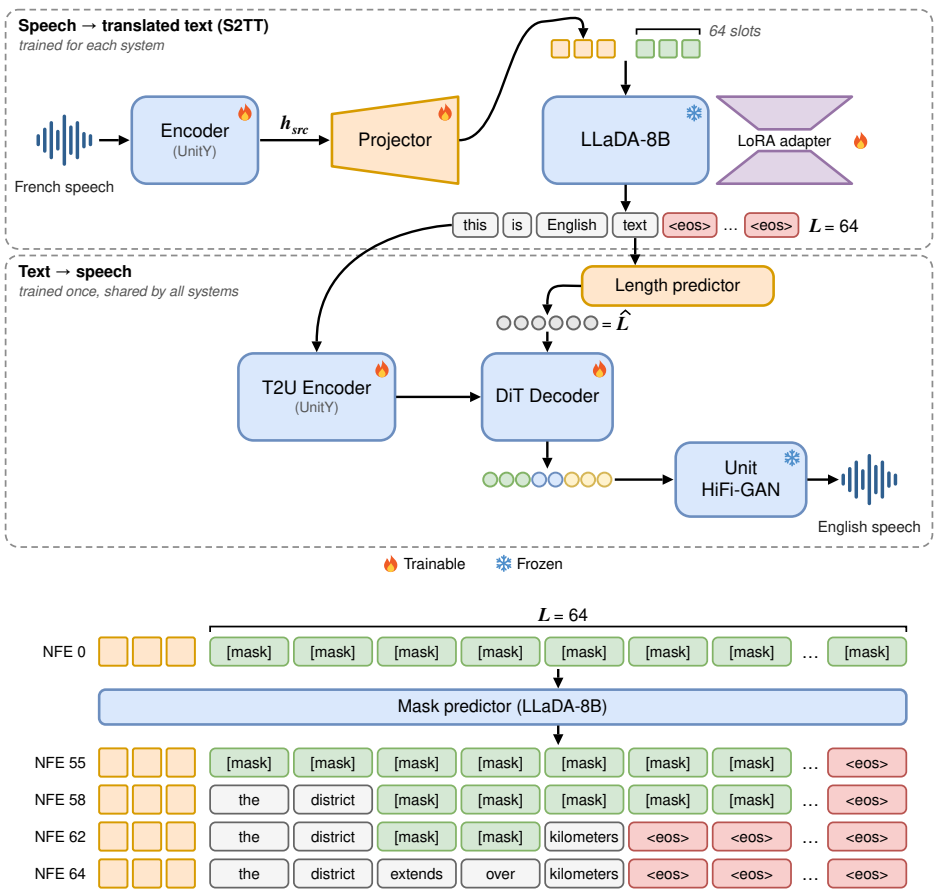

Figure 1: The LLaDA system (top) and the decoding of one test sentence (bottom). Top: in the speech-to-translated-text (S2TT) half, the speech encoder and the projector turn the French speech into a prefix for LLaDA-8B, which is adapted with LoRA and writes the English text on a canvas of 64 slots; the text-to-speech half, shared by all systems (Section 3.1), encodes the text with the T2U encoder, sets the number of units $\hat{L}$ with the length predictor, generates the units with the DiT decoder and renders them into speech with the unit HiFi-GAN vocoder. The two halves are trained separately; the icons refer to each half's own training. Bottom: the canvas after 0, 55, 58, 62 and 64 decoder calls (NFE), with its first seven slots and its last one; orange squares stand for the speech prefix, green slots are masked and red slots hold EOS. Each call commits one slot, and the output is "the district extends over kilometers".

itself decides where the translation ends, on a fixed canvas of 64 slots, with no separate text-length predictor. LLaDA reads text only, so we give it speech through a projector into its embedding space, as LLaDA-V does for images (You et al., 2025): a randomly initialised two-layer MLP maps the states of the speech encoder into the space of LLaDA's word embeddings, and the projected states form a prefix in front of the response canvas; no text prompt is used, and the prefix only conditions the model. We make one change before the projector: every four consecutive encoder states are stacked into one vector, the concatenation downsampling of SLAM-ASR (Ma et al., 2024).

Training. The target is the English transcript in LLaDA's tokenizer, padded with EOS to the 64-slot canvas; the prefix is never masked, and the loss, LLaDA's supervised fine-tuning objective without change, is the cross-entropy on the masked slots of the response, EOS pads included (Appendix F). LLaDA's weights stay frozen and are adapted with LoRA (Hu et al., 2021); the projector and the speech encoder are trained in full. Training is a single stage of 9,900 updates (2.91 epochs), and Appendix G lists every hyper-parameter. The variable-length extensions of Section 3.3 each start from this model and receive one further epoch of adaptation. To control for that extra epoch,

this model also receives one further epoch of its own objective, with the same number of updates and the same source-frame batch budget as the extensions, the adapter folded into the weights and a fresh adapter trained; we call this the continued model.

3.3 VARIABLE-LENGTH EXTENSIONS OF LLADA: FLEXMDM AND DREAMON

LLaDA decides where the translation ends inside a canvas of fixed size. The two extensions of this section change the canvas itself: FlexMDM grows it from nothing, and DreamOn expands or contracts it while decoding. Both start from the LLaDA model of Section 3.2 with its adapter folded into the weights, which then stay frozen, and each receives one additional epoch of training, 3,970 updates, with its own objective, through new LoRA adapters (Appendix G); the frame stacking is retained, and the speech encoder and the projector continue to train from their values in that model.

3.3.1 FLEXMDM

Model. FlexMDM (Kim et al., 2025) generates a sequence of variable length: it starts from an empty sequence and repeatedly inserts mask tokens and unmasks them. Besides the unmasking head, an insertion head predicts, for every gap of the current sequence, the expected number of tokens still to be inserted there. We retrofit the LLaDA model as the authors retrofit LLaDA (Kim et al., 2025, App. F.3), adding a time embedding and the insertion head (Appendix F). The sequence starts empty and can grow to $L = 64$ tokens. At decoding the paper turns the predicted expectations into Poisson draws of new masks; we replace the draws by a deterministic credit (Appendix B.2).

Training. The target is the English transcript without an EOS token, cut to 64 tokens if longer. Every target token draws an insertion time s with distribution function a and a later unmasking time; at the sampled time t_b, tokens not yet inserted are removed, tokens inserted but not yet unmasked are replaced by masks, the remaining tokens are packed to the left, and the number of removed tokens in each gap is the target of the insertion head. Besides the schedules below, three changes to the paper's recipe are ours: how the unmasking times are drawn, a sampling of t_b that sees early times more often, and sorting the insertion times of some examples along the position (Appendix F). The loss adds FlexMDM's cross-entropy on the masked tokens and its loss on the predicted gap counts, each weighted by its schedule's rate factor, with our own floor on the insertion weight (Appendix F).

Schedules. Decoding never removes or re-masks a token, so a word committed while tokens are still missing from the sequence cannot be revised. We therefore choose our own insertion and unmasking schedules, which favour early insertion and delay token commitments (Appendix F).

3.3.2 DREAMON

Model. DreamOn (Wu et al., 2026) is a masked diffusion method for variable-length generation that was developed for code infilling and built on the diffusion language models Dream-Coder-7B and DiffuCoder-7B. It uses two length-control states of the vocabulary, an expand state and a delete state: a mask decoded as the expand state becomes two masks, and a mask decoded as the delete state is removed from the sequence, so the canvas grows or shrinks while it is being filled. Both states are predicted like ordinary tokens, and the architecture of the diffusion language model is unchanged. We apply DreamOn to our speech translation model so that the translation canvas can change its length during decoding. The delete state is the EOS token, as in the authors' code; the expand state, a dedicated token in the authors' tokenizer, is a reserved token of the LLaDA vocabulary in ours; the network is the plain LLaDA model without the time embedding of FlexMDM.

Training. Training uses the DreamOn objective: a random number e_b of delete states, at most six, is appended to the target transcript of n_b tokens, where the paper appends from 0 to 64 (Wu et al., 2026, Sec. 4.1); each transcript token is masked independently with a probability r_b drawn uniformly from $[0, 1]$ per example b, and every appended delete state is masked. For each masked transcript position whose right neighbour is also masked, a draw with a merging probability (Appendix F) decides, as in the authors' code, whether the neighbour is dropped from the input and the position's prediction target becomes the expand state. The model reads the shortened canvas and learns to predict the transcript token, the expand state or the delete state that belongs at each masked position. The transcript is not padded with EOS targets to the full canvas as in Section 3.2; in this epoch the

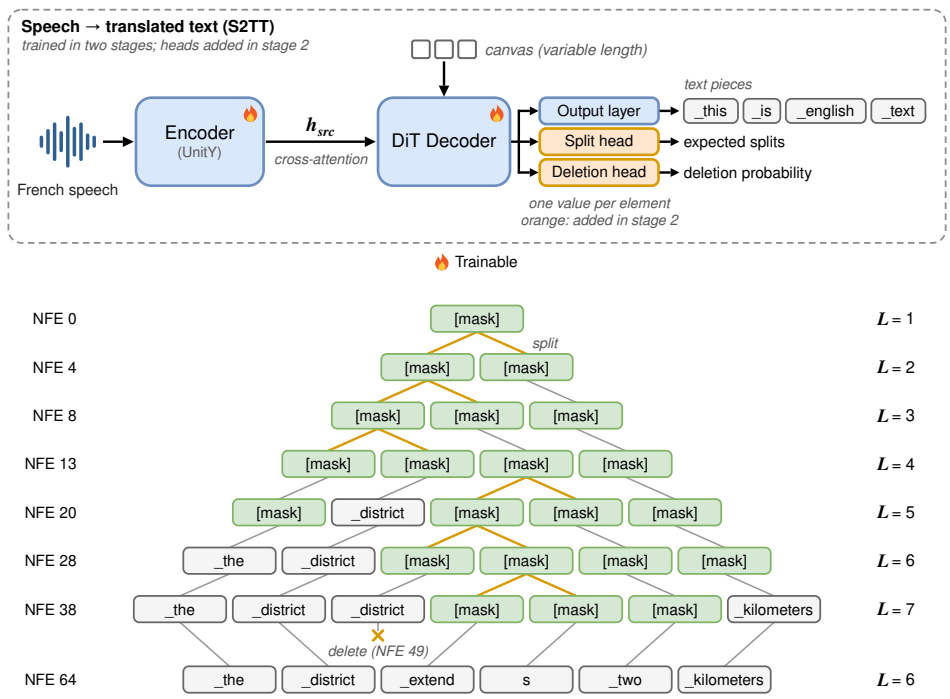

Figure 2: The Branching Flows system (top) and the decoding of the same test sentence as in Figure 1 (bottom). Top: in the speech-to-translated-text (S2TT) half, the DiT decoder cross-attends to the states of the speech encoder and gives, for every element of the canvas, a distribution over the text pieces and the mask token (output layer), an expected number of splits (split head) and a deletion probability (deletion head); the two heads, in orange, are added in the second training stage. The text-to-speech half is the shared one of Figure 1. Bottom: decoding starts from a single masked element and runs 64 steps; the rows show the canvas at the start, after each step with a split and at the end, labelled by the number of decoder calls (NFE), with its length L on the right. An orange fork is a split, which inserts a copy of the element directly after it; grey lines follow the other elements from row to row; the orange cross is the deletion of the second `_district`, after 49 calls. An underscore marks the start of a word, and the output is "the district extends two kilometers".

only EOS targets are the appended delete states. The loss, a weighted cross-entropy on the masked positions, counts the appended delete states of a batch together as one position (Appendix F).

Decoding. Decoding needs an initial number of masks. The authors start from a fixed number; we start from the prediction of a small count head, rounded and clipped to between 1 and 64. The count head is trained after the DreamOn model, on the training set, with the DreamOn model's speech encoder frozen (Appendix F). The predicted length only seeds the canvas; expansions and deletions change it during decoding, as described in Appendix B.3.

3.4 Branching Flows

Model. The three systems above all build on a language model pretrained on large amounts of text. Our last system studies variable-length generation in the other regime: a small decoder of the Dub-S2ST decoder architecture (Choi et al., 2025) that we train ourselves, without a pretrained large language model, using Branching Flows (Billera et al., 2025). Branching Flows was developed for generation in which the number of elements is not known in advance, and it was demonstrated on small molecules, antibody sequences and protein structures. In our speech-conditioned version every element of the canvas can split into copies or be deleted during generation, so the length of the output changes through repeated split and deletion predictions conditioned on the speech. The speech encoder is the UnitY encoder of Section 3.1, loaded from the released checkpoint and fine-tuned in both training stages; the decoder cross-attends to its states and writes the English transcript

as pieces of the UnitY text vocabulary, and the same text-to-unit stage as for the other systems then turns the transcript into non-deduplicated units. The decoder is a new model with the architecture of Section 3.1, absolute position embeddings included, trained from scratch except for its text-piece embedding table, which is copied from a UnitY text decoder that we had fine-tuned (Appendix H). For each element the decoder gives a distribution over the text vocabulary and the mask token, and two heads give its split and deletion predictions.

Split and deletion heads. The split head predicts the expected number $\hat{\mu}$ of further splits of the element by the end of generation, and the deletion head the probability $\hat{\rho}$ that it is deleted by then (Appendix H). At decoding, the authors' sampler turns $\hat{\mu}$ and $\hat{\rho}$ into a Poisson number of splits and a Bernoulli deletion at each step, and we replace both draws by running credits (Appendix B.4).

Training. Training has two stages. In the first stage the transcript is corrupted by masking each piece independently with probability $1 - t$, for a time t drawn uniformly, and the decoder is trained with a label-smoothed cross-entropy on the masked positions. The first-pass text decoder of UnitY is kept as an auxiliary autoregressive decoder on the same encoder states; its cross-entropy is added with weight one during training, and it is not used at test time. We keep the first-stage checkpoint with the highest development BLEU when decoded on a fixed canvas of the target length, one masked position per piece of the target transcript. The second stage starts from the first-stage weights of the encoder, the decoder and the auxiliary decoder, all of which remain trainable; the output layer gains a row for the mask token, starting at zero, and the decoder with its two heads is trained with the Branching Flows objective. Each training example is built from a transcript: copies of randomly chosen pieces are inserted next to their originals, each copy bringing one deletion mark, and the pieces and copies, in order, become the leaves of a binary tree (Billera et al., 2025, App. A.3), whose split times are drawn from the root down and whose marked leaves are removed at random times after their parents' splits. At a training time t the canvas holds the elements present at t, each standing for one leaf or, before its split, for a set of leaves, with a token state obtained by running the discrete base process down the tree; an element is trained to predict its anchor, the text piece of a single leaf and otherwise the mask token (Billera et al., 2025, App. A.4), its number of further splits, and whether it is a single marked leaf (Appendix H). The second-stage loss averages over the canvas elements a label-smoothed cross-entropy against the anchor and the losses of the two heads, each weighted by $1/(1.2 - t)$, and adds the auxiliary cross-entropy as in the first stage; the label smoothing and the auxiliary term are ours. The second stage ran for 17,760 updates (60 epochs), and we use its last checkpoint; Appendix I lists the hyper-parameters of both stages.

Decoding. At test time the canvas starts as a single masked element and is evolved over 64 steps of equal length, with one decoder call per step. Each step first updates the token states with a confidence-based rule of our own and then applies the splits and deletions allotted by the running credits; the expected number of splits of every element is multiplied by $m = 1.2$, the value with the highest BLEU among 1.0, 1.1, 1.2 and 1.3 on the first 500 development utterances. Figure 2 shows the system and the decoding of one test sentence, and Appendix B.4 gives the decoding in full.

4 Results and analysis

We evaluate every system on the full CVSS-C French–English test set of 14,759 utterances. LLaDA, FlexMDM, DreamOn and Branching Flows render their text into speech with the same text-to-unit model and vocoder (Section 3.1), and the speech of every row that we decode, the baselines included, is transcribed by the same ASR model, so ASR-BLEU is the one measure shared by all rows. When a text decoder returns an empty string, the text-to-unit stage renders the fixed word "yes" in its place, so that every utterance has audio; this affects only the speech, and so only ASR-BLEU and its brevity penalty, since text BLEU and the number of empty texts are measured on the output of the text decoder. The systems of Table 1 build on the pretrained LLaDA-8B-Base language model and those of Table 2 do not, so we compare systems only within each table. Each row is one decode of one checkpoint; the three Branching Flows rows decode the same checkpoint.

Systems built on LLaDA. LLaDA, which decodes on a fixed canvas, reaches 28.10 ASR-BLEU with a brevity penalty of 0.866. FlexMDM and DreamOn, which change the size of the canvas while decoding, reach 29.15 and 29.86, with brevity penalties of 1.000 and 0.993. All three receive

Table 1: Systems built on LLaDA-8B-Base, on the CVSS-C French–English test set (14,759 utterances). Text BLEU scores the text written by the text decoder; ASR-BLEU and its brevity penalty (BP) score the synthesised speech. All three rows start from the LLaDA model after 9,900 updates (2.91 epochs) and are trained for one further epoch (3,970 updates): the LLaDA row with its own objective (the continued model of Section 3.2), FlexMDM and DreamOn with their extensions.

System	Text BLEU	Empty texts	ASR-BLEU	BP
LLaDA	31.46	1,060	28.10	0.866
FlexMDM	32.51	0	29.15	1.000
DreamOn	33.49	173	29.86	0.993

Table 2: Baselines and Branching Flows on the CVSS-C French–English test set (14,759 utterances). Decoding is autoregressive (AR) or non-autoregressive (NAR). S2UT and Dub-S2ST are the checkpoints of the Dub-S2ST authors (Choi et al., 2025), and UnitY is the model released with StreamSpeech whose encoder initialises ours (Section 3.1); we decode all three, while the scores marked ‡ are reported from the original papers. For Branching Flows, m is a fixed number that multiplies every expected split count at every step (Section 3.4). The last row instead rescales the expected split counts at every step toward a target length computed from the reference translation, so it is a diagnostic, not a system. Our system is in bold.

System	Decoding	ASR-BLEU	BP
S2UT	AR	23.54	0.967
TranSpeech‡ (Huang et al., 2023)	NAR	18.03	–
DiffNorm‡ (Tan et al., 2024)	NAR	19.53	–
Dub-S2ST	NAR	23.35	0.966
UnitY	AR	25.39	0.991
Branching Flows, $m = 1.0$	NAR	22.94	0.913
Branching Flows, $m = 1.2$	**NAR**	**24.19**	**0.951**
Branching Flows, target-length rescaling (diagnostic)	NAR	24.92	0.977

the same number of updates with the same source-frame batch budget (Section 3.2); DreamOn also uses a count head that is trained separately (Section 3.3.2). The LLaDA and FlexMDM text decoders make 64 calls per utterance, and DreamOn makes 79.14 on average; the text-to-unit decoder adds 64 calls in every system. DreamOn uses all 64 expansions it is allowed on 7,815 of the 14,759 test utterances (53.0%), and these account for 92.2% of its decoder calls (Appendix B.3).

Empty outputs. LLaDA returns 1,060 empty texts. Its text is the canvas up to the first EOS, so an EOS committed in the first slot gives an empty text; among 120 test utterances that we decoded again with a per-step trace, this accounts for all six empty texts. FlexMDM returns no empty text. DreamOn can delete slots (Section 3.3.2 and Appendix B.3), and its 173 empty texts are utterances in which every slot was deleted: in each of them, the number of deletions equals the initial length plus the number of expansions.

Baselines and Branching Flows. Among the baselines, the autoregressive UnitY reaches 25.39 ASR-BLEU, above S2UT (23.54) and the non-autoregressive Dub-S2ST (23.35). Branching Flows reaches 22.94 with $m = 1.0$, 0.41 below Dub-S2ST, and 24.19 with $m = 1.2$, above both Dub-S2ST and S2UT and 1.20 below UnitY. Its text decoder makes 64 calls per utterance.

Where Branching Flows falls short. The length of a Branching Flows output is set during decoding by its splits and deletions (Section 3.4). With $m = 1.0$, the outputs fall further short of the reference as the reference gets longer: on average they are 0.14 pieces longer than the reference for references shorter than 8 pieces, and 0.80, 2.46 and 5.56 pieces shorter for references of 8–13, 14–19 and at least 20 pieces. The fixed multiplier $m = 1.2$, chosen on the development set (Section 3.4), makes the outputs longer on average in each of these four groups. On the test set this helps long references and hurts short ones: for references of at least 20 pieces the outputs are now 4.50 pieces short and the text BLEU of this group rises from 18.3 to 19.8, while for references shorter than 8 pieces the outputs are now 0.37 pieces too long and the text BLEU falls from 31.2 to 30.0. Overall, ASR-BLEU rises from 22.94 to 24.19 (paired bootstrap resampling with 1,000 resamples,

$p = 0.001$) and the brevity penalty from 0.913 to 0.951. One fixed number for all utterances thus trades the short references against the long ones.

Rescaling toward a target length. To see how Branching Flows does when the length of its output is close to right, we decode once more with a target length computed from the reference translation, as a diagnostic. The target length is $N = 1.0718$ times the number of pieces in the reference translation. This decode uses $m = 1.0$ and changes only the predicted numbers of further splits $\hat{\mu}_i$; the deletion and token-update rules are unchanged. At every step, we first count how many elements are still missing: the target length minus the expected number of current elements that will not be deleted, $R = \max(0, N - \sum_j (1 - \hat{\rho}_j))$, with the sum over the current elements. We then share R among the current elements in proportion to their $\hat{\mu}_i$, so that element i uses $R\,\hat{\mu}_i / \sum_j \hat{\mu}_j$ in place of $\hat{\mu}_i$; if every $\hat{\mu}_j$ is zero, each of the L current elements gets R/L. In the last eight of the 64 steps, those that start at $s \geq 0.875$, R is set to 0, so no further splits are made. So where $m = 1.2$ multiplies every $\hat{\mu}_i$ by 1.2, the rescaling multiplies them by $R / \sum_j \hat{\mu}_j$, a number that is recomputed at every step and differs from one utterance to the next. The value 1.0718 comes from the first 500 development utterances, decoded with the same rule but with targets predicted by a separately fitted length predictor instead of taken from the reference: there the outputs were in total 0.933 times as long as their targets, and $1/0.933 = 1.0718$. This decode reaches 24.92 ASR-BLEU, up from 24.19 with $m = 1.2$ ($p = 0.001$), with a brevity penalty of 0.977, and the gap to the autoregressive UnitY shrinks from 1.20 ASR-BLEU with $m = 1.2$ to 0.47. Besides the length, the rescaling changes when the elements split and stops the late splits, so this comparison does not isolate the effect of length; it does suggest that length control is an important limitation of Branching Flows. The length errors are largest for long references: even with the rescaling, the outputs for references of at least 20 pieces remain 1.59 pieces short on average.

5 Conclusion

We studied how non-autoregressive speech-to-speech translation can decide the length of its output, with two kinds of text decoders in the two-pass layout of UnitY. The first kind feeds speech to the pretrained masked diffusion language model LLaDA-8B-Base through a projector and fine-tunes it on speech translation for fewer than four epochs. On a fixed canvas, LLaDA reaches 28.10 ASR-BLEU; its two variable-length extensions, FlexMDM, which grows the canvas from empty by inserting masks, and DreamOn, which expands or contracts the canvas while decoding, reach 29.15 and 29.86 with the same number of additional updates. The second kind works without a pretrained large language model: a small decoder trained with Branching Flows starts from a single element, and elements split into copies or are deleted during decoding, so the length of the translation is decided on the way. With a rate multiplier chosen on the development set, it reaches 24.19 ASR-BLEU, above the non-autoregressive Dub-S2ST (23.35) and the autoregressive S2UT (23.54), and 1.20 below the autoregressive UnitY (25.39).

Limitations. The LLaDA-based systems depend on a model pretrained on large amounts of text, which we adapt to translate speech; we did not isolate how much of their translation quality comes from this pretraining. For Branching Flows, length control is an important limitation: its outputs are too short for long references, and when we rescale the decode toward a target length computed from the reference translation, as a diagnostic, ASR-BLEU rises from 24.19 to 24.92 (Section 4). Future work could improve length control for long translations through better split and deletion predictions or a better decoding rule.

AI use statement

In this work, we used generative AI tools to implement methods: we used Claude (Anthropic) as a coding agent. We have not used generative AI tools to generate synthetic data sets, develop theoretical models or conceptual frameworks, propose or refine hypotheses, or assist with translation, and formulating or proving mathematical claims and qualitative or thematic data analysis are not applicable to this work. Additionally, we used Claude to search for and summarise related literature, to draft parts of the text and to create the figures, and ChatGPT (OpenAI) to format the manuscript and refine its wording. We have reviewed all AI-assisted work: new code was checked by reproducing

previously recorded results before use, every reported number was taken from the output of the run that produced it, and every statement about prior work was checked against the original paper or its released code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *arXiv preprint arXiv:2107.03006*, 2021.

Arun Babu, Changhan Wang, Andros Tjandra, Kushal Lakhotia, Qiantong Xu, Naman Goyal, Kritika Singh, Patrick von Platen, Yatharth Saraf, Juan Pino, Alexei Baevski, Alexis Conneau, and Michael Auli. XLS-R: Self-supervised cross-lingual speech representation learning at scale. In *Interspeech 2022*, pp. 2278–2282, 2022.

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In *Advances in Neural Information Processing Systems*, volume 33, pp. 12449–12460, 2020.

Lukas Billera, Hedwig Nora Nordlinder, Jack Collier Ryder, Anton Oresten, Aron Stålmarck, Theodor Mosetti Björk, and Ben Murrell. Branching flows: Discrete, continuous, and manifold flow matching with splits and deletions. *arXiv preprint arXiv:2511.09465*, 2025.

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. In *International Conference on Machine Learning (ICML)*, 2024.

Sanyuan Chen, Chengyi Wang, Zhengyang Chen, Yu Wu, Shujie Liu, Zhuo Chen, Jinyu Li, Naoyuki Kanda, Takuya Yoshioka, Xiong Xiao, Jian Wu, Long Zhou, Shuo Ren, Yanmin Qian, Yao Qian, Jian Wu, Michael Zeng, Xiangzhan Yu, and Furu Wei. WavLM: Large-scale self-supervised pre-training for full stack speech processing. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1505–1518, 2022.

William Chen, Wangyou Zhang, Yifan Peng, Xinjian Li, Jinchuan Tian, Jiatong Shi, Xuankai Chang, Soumi Maiti, Karen Livescu, and Shinji Watanabe. Towards robust speech representation learning for thousands of languages. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024a.

Yushen Chen, Zhikang Niu, Ziyang Ma, Keqi Deng, Chunhui Wang, Jian Zhao, Kai Yu, and Xie Chen. F5-TTS: A fairytaler that fakes fluent and faithful speech with flow matching. *arXiv preprint arXiv:2410.06885*, 2024b.

Jeongsoo Choi, Jaehun Kim, and Joon Son Chung. Dub-S2ST: Textless speech-to-speech translation for seamless dubbing. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025.

Qingkai Fang, Yan Zhou, and Yang Feng. DASpeech: Directed acyclic transformer for fast and high-quality speech-to-speech translation. In *Advances in Neural Information Processing Systems*, 2023.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *arXiv preprint arXiv:2407.15595*, 2024.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-Predict: Parallel decoding of conditional masked language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *International Conference on Learning Representations (ICLR)*, 2018.

Jiatao Gu, Changhan Wang, and Jake Zhao. Levenshtein transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. arXiv:1905.11006.

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. *arXiv preprint arXiv:2106.09685*, 2021.

Fei Huang, Hao Zhou, Yang Liu, Hang Li, and Minlie Huang. Directed acyclic transformer for non-autoregressive machine translation. In *International Conference on Machine Learning (ICML)*, 2022. arXiv:2205.07459.

Rongjie Huang, Jinglin Liu, Huadai Liu, Yi Ren, Lichao Zhang, Jinzheng He, and Zhou Zhao. TranSpeech: Speech-to-speech translation with bilateral perturbation. In *International Conference on Learning Representations (ICLR)*, 2023.

Hirofumi Inaguma, Sravya Popuri, Ilia Kulikov, Peng-Jen Chen, Changhan Wang, Yu-An Chung, Yun Tang, Ann Lee, Shinji Watanabe, and Juan Pino. UnitY: Two-pass direct speech-to-speech translation with discrete units. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

Ye Jia, Ron J. Weiss, Fadi Biadsy, Wolfgang Macherey, Melvin Johnson, Zhifeng Chen, and Yonghui Wu. Direct speech-to-speech translation with a sequence-to-sequence model. In *Interspeech*, 2019.

Ye Jia, Michelle Tadmor Ramanovich, Quan Wang, and Heiga Zen. CVSS corpus and massively multilingual speech-to-speech translation. In *Language Resources and Evaluation Conference (LREC)*, 2022.

Jaeyeon Kim, Lee Cheuk-Kit, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngotiaoco, Sitan Chen, and Michael Albergo. Any-order flexible length masked diffusion. *arXiv preprint arXiv:2509.01025*, 2025.

Matthew Le, Apoorv Vyas, Bowen Shi, Brian Karrer, Leda Sari, Rashel Moritz, Mary Williamson, Vimal Manohar, Yossi Adi, Jay Mahadeokar, and Wei-Ning Hsu. Voicebox: Text-guided multilingual universal speech generation at scale. In *Advances in Neural Information Processing Systems*, 2023.

Ann Lee, Peng-Jen Chen, Changhan Wang, Jiatao Gu, Sravya Popuri, Xutai Ma, Adam Polyak, Yossi Adi, Qing He, Yun Tang, Juan Pino, and Wei-Ning Hsu. Direct speech-to-speech translation with discrete units. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning (ICML)*, 2024.

Ziyang Ma, Guanrou Yang, Yifan Yang, Zhifu Gao, Jiaming Wang, Zhihao Du, Fan Yu, Qian Chen, Siqi Zheng, Shiliang Zhang, and Xie Chen. An embarrassingly simple approach for LLM with strong ASR capacity. *arXiv preprint arXiv:2402.08846*, 2024.

Shivam Mehta, Ruibo Tu, Jonas Beskow, Éva Székely, and Gustav Eje Henter. Matcha-TTS: A fast TTS architecture with conditional flow matching. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

William Peebles and Saining Xie. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, 2024.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data. In *Advances in Neural Information Processing Systems*, 2024.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning (ICML)*, pp. 5976–5985, 2019. PMLR 97.

Weiting Tan, Jingyu Zhang, Lingfeng Shen, Daniel Khashabi, and Philipp Koehn. DiffNorm: Self-supervised normalization for non-autoregressive speech-to-speech translation. In *Advances in Neural Information Processing Systems*, 2024.

Zirui Wu, Lin Zheng, Zhihui Xie, Jiacheng Ye, Jiahui Gao, Shansan Gong, Yansong Feng, Zhenguo Li, Wei Bi, Guorui Zhou, and Lingpeng Kong. DreamOn: Diffusion language models for code infilling beyond fixed-size canvas. In *International Conference on Learning Representations (ICLR)*, 2026.

Zebin You, Shen Nie, Xiaolu Zhang, Jun Hu, Jun Zhou, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. LLaDA-V: Large language diffusion models with visual instruction tuning. *arXiv preprint arXiv:2505.16933*, 2025.

Shaolei Zhang, Qingkai Fang, Shoutao Guo, Zhengrui Ma, Min Zhang, and Yang Feng. StreamSpeech: Simultaneous speech-to-speech translation with multi-task learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2406.03049.

Yongxin Zhu, Zhujin Gao, Xinyuan Zhou, Zhongyi Ye, and Linli Xu. DiffS2UT: A semantic preserving diffusion model for textless direct speech-to-speech translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

A THE SILENCE-PADDING PILOT

Before the main systems, we ran a single-pass pilot in the S2UT setting (Lee et al., 2022), speech straight to units with no text in between, to see whether a decoder on a fixed canvas can decide its own length. We first looked for the unit that stands for silence: we rendered candidate units through the vocoder, kept those that come out silent, and took the one that most often ends a training target. Each training target is then the non-deduplicated unit sequence of the utterance followed by this silence unit up to a canvas of 400 slots, a length picked from the length distribution of the training targets so that almost every utterance fits; the few longer ones are dropped from training and truncated at test time. The decoder is a Dub-S2ST-style diffusion transformer with absolute position embeddings added, on the shared encoder, trained for 30k updates with the masked cross-entropy of Dub-S2ST: the target is masked at a rate drawn at random for each example, and the loss is the cross-entropy on the masked positions, silence slots included. At test time all 400 slots are decoded in 64 steps by re-masking the least confident predictions at every step; the units go to the unit HiFi-GAN vocoder without duration prediction, since they are not deduplicated, and the audio is scored by ASR-BLEU like every other system. Decoded plainly, the pilot reaches 15.79 ASR-BLEU with a brevity penalty of 0.728: the translations come out too short. A decode-time change alone repairs this: when the silence unit is excluded from the confidence ranking, so that content is committed first and silence fills whatever remains, the same model reaches 21.77 with a brevity penalty of 0.969. LLaDA's authors report the same symptom and a corresponding remedy for their EOS-padded fine-tuning: under pure diffusion sampling the proportion of EOS tokens becomes very high, generations become extremely short, and they set the confidence of EOS to zero (Nie et al., 2025). The pilot thus showed that a fixed canvas padded with an end marker can carry the length decision, and that its decoding needs care. The LLaDA system of Section 3.2 pads its training targets in the same way, with EOS in place of silence; we present it without any decode-time rule, so that it can be compared with the variable-length extensions on an equal footing.

B DECODING OF EACH SYSTEM

B.1 LLADA

Decoding is LLaDA's low-confidence remasking on the fixed canvas (Nie et al., 2025). All 64 slots start masked. At each of 64 calls the model reads the speech prefix and the canvas and predicts every masked slot in parallel. Only the most confident of these predictions is committed, one token per call; the others are masked again, which is the remasking of LLaDA, and a committed slot is never changed. The confidence of a slot is the largest predicted token probability at that slot, and the token committed there is that most probable token. The output is cropped at the first EOS.

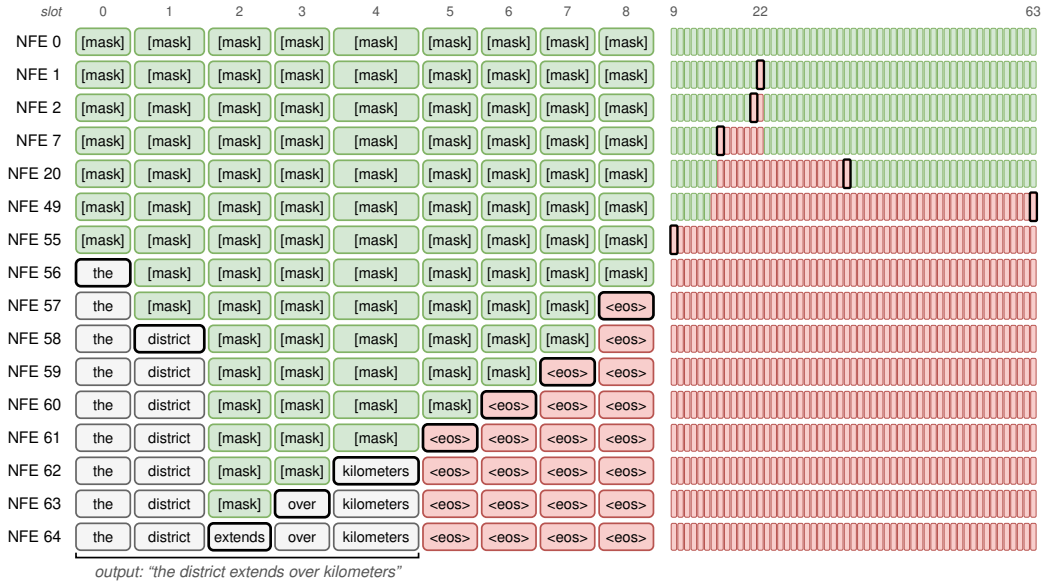

Figure 3: LLaDA decoding of one test sentence, call by call. Each row is the canvas of 64 slots after the given number of decoder calls (NFE); slots 0 to 8 are drawn with their token and slots 9 to 63 as small cells. Green slots are masked, red slots hold EOS and grey slots hold words; a black outline marks the slot committed by that row's call. The speech prefix, which the model reads at every call, is not drawn.

Figure 3 shows this decoding on one test sentence with the model of the LLaDA row of Table 1; the reference translation is "the district extends for kilometers". The first 55 calls all commit EOS, each with a confidence of at least 0.99: the first EOS goes to slot 22, and the EOS then spread outward until they fill slots 9 to 63. The remaining nine slots then receive the five words of the output and four more EOS, and the last call commits "extends" with a confidence of 0.48. The first EOS is in slot 5, so the output is "the district extends over kilometers".

B.2 FLEXMDM

The model, its insertion head and the schedules a and β are those of Section 3.3.1 and Appendix F. Decoding starts at $t = 0$ from an empty sequence behind the speech prefix and makes 64 calls, the time advancing by $1/64$ per call. In every call but the last, the number of masks to commit is a Poisson draw with mean equal to the number of masks times $\dot{\beta}(t)/(1 - \beta(t))$ times $1/64$, as in the paper's adaptive inference (Kim et al., 2025), capped at the number of masks; the masks with the largest margin between their two most likely tokens are committed to their most likely token. The paper ranks masks by their largest token probability (Kim et al., 2025, App. E.1); margin ranking is the default of the code the FlexMDM authors released for their LLaDA retrofit. Where the paper then draws, for each gap, a Poisson number of new masks with mean equal to the predicted expectation times $\dot{a}(t)/(1-a(t))$ times $1/64$, we insert deterministically while the unmasking count

stays random. In each call the means of the $n + 1$ gaps are added to a running credit C, and $k = \min(\lfloor C \rfloor, n + 1)$ masks are inserted, one into each of the k gaps with the largest predicted expectations. Then $C \leftarrow C - k$, so unused credit, which can exceed one when the cap $n + 1$ binds, carries over to the next call. If the insertion would take the sequence beyond 64 tokens, nothing is inserted for that example in that call, as in the authors' released sampler, and its credit is reset to zero. The last call fills every remaining mask with its most likely token and inserts nothing. Nothing is removed from the canvas during decoding and it never shrinks: when the model writes an EOS token into a mask, the token stays in place and is dropped only when the final canvas is turned into text, so it neither ends the output nor shortens the canvas.

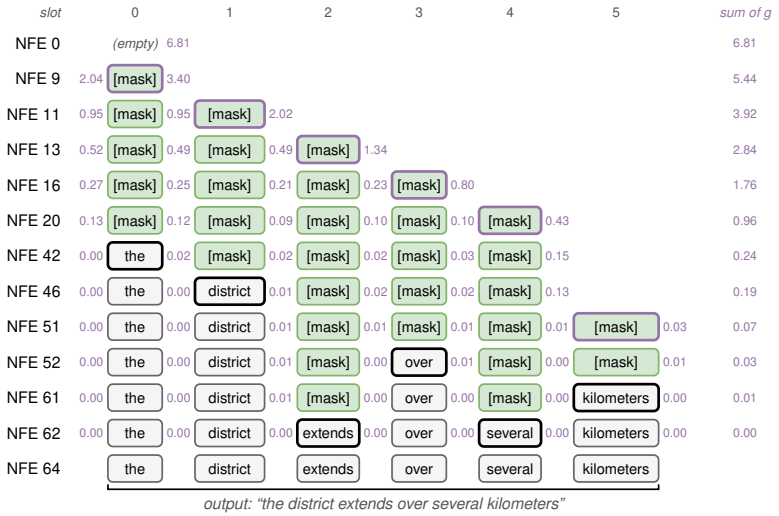

Figure 4: FlexMDM decoding of the test sentence of Figure 3, call by call. Each row is the sequence after the given number of decoder calls (NFE); rows are shown for the start, the end and every call that changed the sequence, which starts empty. The purple numbers are the insertion head's expected number of tokens still to be inserted in each gap of that sequence, as read by the next call, and their sum is given on the right. Green slots are masked and grey slots hold words; a purple outline marks the mask inserted by that row's call and a black outline the slots it committed. The speech prefix, which the model reads at every call, is not drawn.

Figure 4 shows this decoding on the test sentence of Figure 3 with the model of the FlexMDM row of Table 1. On the empty sequence the insertion head expects 6.81 tokens in its one gap, but $\dot{a}(t)/(1 - a(t))$ is small near $t = 0$, so the credit gains only 0.0165 in the first call and first reaches one in call 9, which inserts the first mask. Calls 11, 13, 16 and 20 insert one mask each, always into the last gap, where the expectation is largest; on the sequences with zero to five masks the expectations add up to 6.81, 5.44, 3.92, 2.84, 1.76 and 0.96, close to the number of output tokens not yet in the sequence. Commits start late, as $\beta(t) = t^3$ keeps their expected number per call small: 0.02 in call 21, with five masks, and 0.13 in call 42, whose draw commits the first word, "the"; "district" follows in call 46. Call 51 inserts the sixth mask, again into the last gap, and calls 52, 61 and 62 commit "over", "kilometers", and "extends" and "several" together; calls 63 and 64 change nothing. No EOS is written, so the output is the whole sequence, "the district extends over several kilometers".

B.3 DreamOn

The model, its expand and delete states and the count head that sets the initial number of masks are those of Section 3.3.2 and Appendix F. Each call predicts every mask and commits the single mask with the highest top probability to its most likely token, which can be an ordinary token, the expand state or the delete state; the authors' code also commits one mask per call. A committed expand state becomes two masks: its position becomes a mask again and a new mask is inserted directly to its right. When a delete state is committed, every mask to its right first becomes a delete state as

well, a rule the authors call broadcasting deletion; the delete states are then removed and the tokens after them move left to close the gaps, so when only masks lie to the right, the canvas ends where the delete state was committed. The paper states the rule for that case (Wu et al., 2026, Sec. 3.4); we follow the authors' code, which also applies it when committed tokens lie to the right of the delete state and preserves those tokens. The expand state is blocked when the canvas already holds 64 positions, masks included, or when 64 expansions have been made; the authors' code blocks it under the same two conditions with its own maximum length. Decoding stops when no mask remains; if a 192nd call is reached, the expand state is blocked on that call and every remaining mask is committed, so decoding ends there. The paper ranks masks by the entropy of their predicted distributions, samples tokens at temperature 0.2 with nucleus filtering at 0.9 and uses a maximum length of 128 (Wu et al., 2026, Secs. 3.3 and 4.1); we rank masks by their top probability, take the most likely token and keep the 64-position canvas of Section 3.2 as the maximum length.

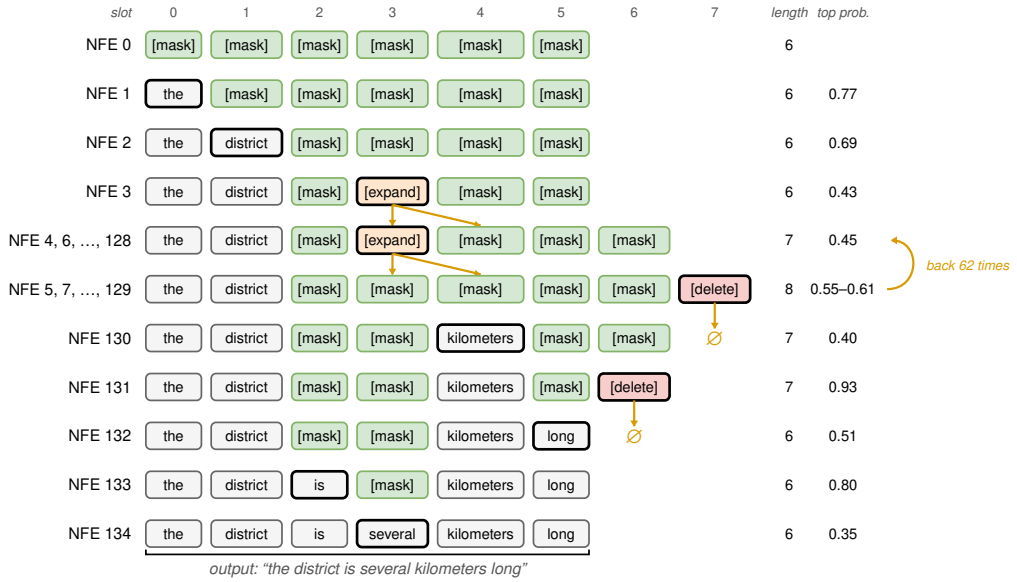

Figure 5: DreamOn decoding of the test sentence of Figure 3, call by call. Each row is the canvas as the given decoder call (NFE) leaves it after its commit, before an expand state becomes two masks and before a delete state is removed; NFE 0 is the initial canvas. The two rows labelled with several calls each stand for 63 calls of a loop of two, and the arrow on the right leads from the second row back to the first. Green slots are masked and grey slots hold words; the expand state is orange and the delete state red, and a black outline marks the slot committed by that row's call. Orange arrows show an expand state becoming two masks and a delete state being removed (∅). The right-hand columns give the canvas length and the probability of the committed token, as a range for the loop rows. The speech prefix, which the model reads at every call, is not drawn.

Figure 5 shows this decoding on the test sentence of Figure 3 with the model of the DreamOn row of Table 1. The count head predicts 6.09 tokens, so the canvas starts with six masks; the reference translation has five LLaDA tokens. Calls 1 and 2 commit "the" and "district", and call 3 commits the expand state in slot 3, which makes the canvas seven slots long. Calls 4 to 129 then alternate between two moves, 63 times each: on the seven-slot canvas the most likely commit is the expand state in slot 3, with probability 0.45, which makes the canvas eight slots long; on that canvas it is the delete state in the last slot, with probability 0.55, which removes that slot and gives back the same seven-slot canvas. Call 128 makes the 64th expansion, so the expand state is blocked from call 129 on, where the delete state is committed with probability 0.61. Calls 130 to 134 then commit "kilometers", a delete state in the last slot, "long", "is" and "several", and the output is "the district is several kilometers long", after 134 calls, 64 expansions and 64 deletions. On the whole test set, 7,815 of the 14,759 utterances (53.0%) also use all 64 expansions, and they account for 92.2% of all decoder calls; the other utterances need 13.06 calls on average.

B.4 BRANCHING FLOWS

The model, its split and deletion heads and the base-process schedules κ_1 and κ_2 are those of Section 3.4 and Appendix H. At test time the canvas starts as a single masked element and is evolved over 64 steps of equal length in t, with one decoder call per step, made at the start s of each step from s to s'. As in the paper, each step first updates the token states and then applies splits and deletions (Billera et al., 2025, Sec. 3.5). The authors' sampler draws all three at random: the token changes from the base process, the number of splits of each element from a Poisson distribution and each deletion from a Bernoulli distribution. On the first 500 development utterances, and without the split multiplier introduced below, replacing the split and deletion draws with the credit allocation described below raised BLEU over the authors' sampler, and replacing in addition the token draw with a confidence-based update of our own raised it further; our decoder uses both replacements. The token update follows the schedules of the base process, with L the current number of elements. Committed pieces whose current probability is below one half are re-masked, lowest first, at most $\mathrm{round}(\kappa_2(s')\,L)$ of them. The number of committed pieces is then raised towards $\mathrm{round}((\kappa_1(s') + \kappa_2(s'))\,L)$ by committing, among the elements that were already masked before the step, the most confident text-piece predictions; the mask row is never committed. At the last step nothing is re-masked and every remaining masked element is committed. The two credits c_{del} and c_{split} start at 0 for each utterance, and one step from s to s' on a canvas of L elements is:

1. Call the decoder once on the canvas at time s.
2. Update the token states as described above.
3. For every element i, compute its expected number of splits $r_i = m\,(s' - s)\,\hat{\mu}_i\,h_{\mathrm{s}}(s)$ and its deletion probability $p_i = \hat{\rho}_i\,(1 - e^{-(s'-s)\,h_{\mathrm{d}}(s)})$, where $h_{\mathrm{s}}(s) = 2/(1-s)$ and $h_{\mathrm{d}}(s) = 1/(1-s)$ are the hazard rates of the split-time and deletion-time distributions, and $m = 1.2$.
4. Add $\sum_i p_i$ to c_{del}, let $k = \lfloor c_{\mathrm{del}} \rfloor$, subtract k from c_{del}, and mark for deletion the k elements with the largest p_i.
5. Add $\sum_i r_i$ over the unmarked elements to c_{split}, let $n = \lfloor c_{\mathrm{split}} \rfloor$ and subtract n from c_{split}; if no element is unmarked, add n back to c_{split}; otherwise hand the n splits to the unmarked elements one at a time in decreasing order of r_i, starting again from the first once each has received one.
6. If every element is marked, unmark one of them chosen uniformly at random.
7. Build the new canvas from the unmarked elements in order, each followed directly by as many copies of its updated state as it received splits.

With $m = 1$, r_i and p_i are the mean split count and the deletion probability of the authors' sampler, and $m = 1.2$ is the value with the highest BLEU among 1.0, 1.1, 1.2 and 1.3 on the first 500 development utterances. Step 6 follows the authors' sampler and is the only random choice of our decoder, which is therefore not strictly deterministic. After the last step, the pieces other than the four special symbols of the vocabulary are detokenised into the transcript. A canvas of more than 4,000 elements would end the decoding of its utterance with an empty output; this was never reached in the reported decodes.

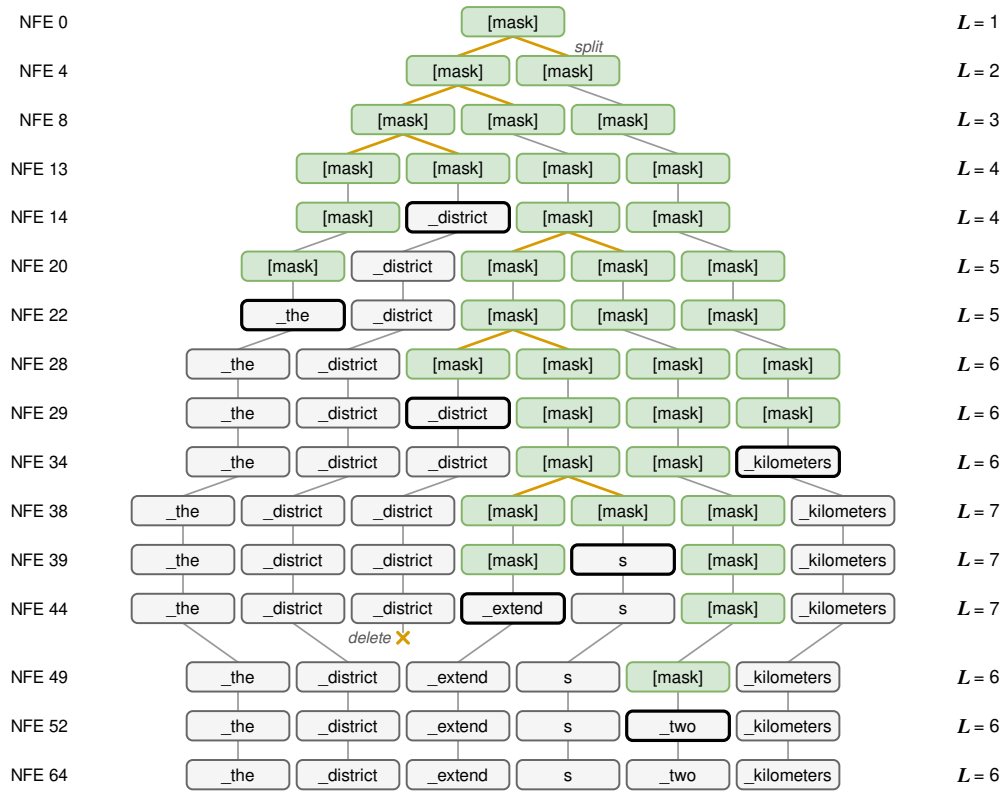

Figure 6: Branching Flows decoding of the test sentence of Figure 3, call by call, drawn as in Figure 2. Decoding starts from a single masked element; each row is the canvas after the given number of decoder calls (NFE), with its length L on the right, and rows are shown for the start, the end and every call that changed the canvas. An orange fork is a split, which inserts a copy of the element directly after it; grey lines follow the other elements from row to row; the orange cross is the deletion of the second `_district`, in call 49. Green elements are masked and grey elements hold text pieces, an underscore marking the start of a word; a black outline marks the piece committed by that row's call. The speech, which the decoder reads through cross-attention at every call, is not drawn.

Figure 6 shows this decoding on the test sentence of Figure 3 with the model of the row "Branching Flows, $m = 1.2$" of Table 2; the reference translation is "the district extends for kilometers". The single masked element adds between 0.26 and 0.28 to the split credit in each of the first four calls, so the credit first reaches one in call 4, which gives the element one copy. The canvas grows by one element in each of calls 4, 8, 13, 20, 28 and 38; each of these splits gives its copy to the element with the largest r_i, always a masked element, and an element that holds a piece has r_i below 0.001 in every call. Pieces are committed from call 14 on: "_district" in call 14, "_the" in call 22, a second "_district" in call 29, "_kilometers" in call 34, "s" in call 39 and "_extend" in call 44. From call 30 to call 49 each of the two "_district" elements has a deletion probability between 0.015 and 0.037 per call, while every other element has at most 0.002; of the deletion credit of 1.02 gathered over the first 49 calls, 0.95 comes from the "_district" elements. The credit reaches one in call 49, which deletes the element with the largest p_i, the second "_district" (0.0366 against 0.0355). Call 52 commits "_two" into the last masked element, and calls 53 to 64 change nothing. The pieces "_extend" and "s" are joined into one word, so the output is "the district extends two kilometers".

C RENDERING TEXT INTO SPEECH

The text-to-unit model. The input of the text-to-unit model of Section 3.1 is the text lowercased, with punctuation removed, and cut into the pieces of UnitY's 6,000-piece unigram text vocabulary. A length predictor sets the number of units $\hat{L}$ from three counts of this text, its n_p pieces, its n_c characters (spaces included) and its n_w words:

$$\hat{L} = \max\big(2,\ \mathrm{round}(2.392\, n_p + 2.432\, n_c - 1.106\, n_w + 10.180)\big). \tag{1}$$

Its four coefficients, rounded here to three decimals, are fitted by ordinary least squares on the 207,364 CVSS-C training utterances, from the reference English transcript to the number of units of its target speech; at test time the same rule is applied to the text written by each system. The pieces are embedded with sinusoidal position embeddings and encoded by a Transformer initialised from UnitY's text-to-unit encoder, and the diffusion transformer of Section 3.1 generates the units on a canvas of $\hat{L}$ slots while cross-attending to the encoded text.

Other ways of turning text into speech. Our text decoders write text, so the speech comes from a separate stage, and we compared several ways of turning the same text into speech (Table 3). The text is the output of the LLaDA system of Table 1 on the full test set, with a text BLEU of 31.46; every route receives this same text, with the placeholder "yes" in place of its 1,060 empty outputs (Section 4), and the speech of every route is transcribed by the same ASR model and scored against the same references. Our first text-to-unit model wrote deduplicated units, in which consecutive repetitions of a unit are collapsed into one, and the unit HiFi-GAN vocoder predicted how long each unit lasts; this route reaches 27.71 ASR-BLEU. Keeping these units, we replaced the vocoder with unit-Matcha, our own model: a Matcha-TTS model (Mehta et al., 2024) that we trained from scratch on LJSpeech recordings to turn their deduplicated mHuBERT units into mel spectrograms, followed by the LJSpeech HiFi-GAN vocoder released with Matcha-TTS; unit-Matcha predicts the duration of each unit itself and raises the score to 28.12, 0.41 above the unit HiFi-GAN. The text-to-unit model used by LLaDA, FlexMDM, DreamOn and Branching Flows throughout the paper (Section 3.1) has the same architecture but was trained on units with consecutive repetitions retained, one unit per 20 ms, so that the units carry their own durations and the unit HiFi-GAN runs with its duration predictor disabled; this route reaches 28.10, 0.39 above the deduplicated route. Since both the training units and the source of the durations change, this compares two complete routes and does not isolate the effect of the duration predictor. So on the same deduplicated units, unit-Matcha does better than the unit HiFi-GAN, but this complete route scores about the same as the route we use: 28.12 against 28.10, a difference of 0.02 that is not statistically significant ($p = 0.29$). The released Matcha-TTS checkpoint for LJSpeech, which reads the text directly and uses no units, reaches 29.17 with the same HiFi-GAN vocoder as unit-Matcha: on the same translations it is 1.07 above the route used in the main experiments. Both Matcha-TTS models decode with 32 steps and a temperature of 0.667. We test each difference with paired bootstrap resampling with 1,000 resamples; all differences named in this section have $p = 0.001$ except the 0.02 between unit-Matcha and the route we use.

Table 3: Turning the same text into speech: the text of the LLaDA system of Table 1 (text BLEU 31.46) on the CVSS-C French–English test set (14,759 utterances). The third row is the stage used by LLaDA, FlexMDM, DreamOn and Branching Flows throughout the paper.

Units from the text-to-unit model	Speech from the units	ASR-BLEU	BP
Deduplicated	Unit HiFi-GAN, predicted durations	27.71	0.863
Deduplicated	Unit-Matcha (ours)	28.12	0.865
Repetitions retained	Unit HiFi-GAN, durations from the units	28.10	0.866
No units: Matcha-TTS reads the text		29.17	0.872

D SELF-SUPERVISED SPEECH ENCODERS

The LLaDA system of Section 3.2 hears the speech through the Conformer encoder of a UnitY model trained on CVSS-C (Section 3.1); here we replace that encoder with a self-supervised speech encoder pretrained on speech in many languages (Table 4). We use XLS-R (Babu et al., 2022),

wav2vec 2.0 pretrained on 436K hours of speech in 128 languages, in its 300M-parameter version, and XEUS (Chen et al., 2024a), pretrained on over one million hours of speech in 4,057 languages. Our source speech is French, so we did not use wav2vec 2.0 itself, pretrained on LibriSpeech and LibriVox audiobooks (Baevski et al., 2020), or WavLM, pretrained on 94K hours of English speech (Chen et al., 2022). Both encoders read the 16 kHz waveform; their convolutional front end stays frozen, the masking and layer dropping of their pretraining are switched off, and all later layers train with the encoder learning rate of Appendix G. The projector receives a weighted sum of the outputs of all layers, 25 for XLS-R, counting its input to the first Transformer layer, and 19 for XEUS, each normalised by a layer normalisation without parameters; the weights are the softmax of logits that start at zero and train with the learning rate of the projector. Both encoders emit one state per 20 ms, half the interval of the UnitY encoder, so we stack eight consecutive states instead of four, and consecutive prefix vectors stay 160 ms apart. Everything else follows the recipe of the model after 9,900 updates in Section 3.2, with the same batches and the same 9,900 updates, each update split over two GPUs; we therefore compare with that model, before the further epoch of the continued model in Table 1. On the full test set, text BLEU rises from 34.38 with the UnitY encoder to 36.37 with XLS-R and 37.84 with XEUS. These three numbers were decoded with two changes to the decoding of Appendix B.1: EOS is excluded at the first slot, and a slot whose most probable token is EOS gets confidence $-\infty$. The encoders also differ in size, with 33.5M trainable parameters in the UnitY encoder against 311.2M in XLS-R and 569.9M in XEUS, so this comparison does not separate the pretraining from the size of the encoder.

Table 4: Speech encoders for the LLaDA system after 9,900 updates: text BLEU on the CVSS-C French–English test set (14,759 utterances). Trainable parameters count the speech encoder only.

Speech encoder	Input	Trainable parameters	Text BLEU
UnitY Conformer	Filterbank features	33.5M	34.38
XLS-R (300M)	Waveform	311.2M	36.37
XEUS	Waveform	569.9M	37.84

E FLOW MATCHING ON CONTINUOUS SPEECH FEATURES

The text-to-unit stage of Section 3.1 generates discrete units; in this experiment the same model generates the continuous speech features behind the units, which are turned into units only after the last step. The features of an utterance are the 768-dimensional outputs of layer 11 of mHuBERT, one every 20 ms, and its units are the indices of the nearest of 1,000 k-means centres. We subtract the mean feature of the training set and divide all 768 dimensions by one common scale, chosen so that the standardised features z have unit variance on average, like the Gaussian noise they are generated from. The model keeps the architecture, the text-encoder initialisation, the training batches, the learning rate and the 20,000 updates of the discrete model; the unit embedding table becomes a linear layer from the 768 features to the width of the transformer, and the output layer predicts 768 numbers per frame instead of scores over the 1,000 units. It is trained with flow matching (Lipman et al., 2022) on the straight path from noise to data: with a time t drawn uniformly from $[0.001, 0.999]$ for each utterance and noise $\epsilon \sim \mathcal{N}(0, I)$ drawn for every frame and dimension, the transformer reads $z_t = (1-t)\,\epsilon + t\,z$ and predicts the velocity $z - \epsilon$, with the squared error averaged over all frames of the batch and all 768 dimensions. The discrete model copies its transformer from a speech-to-unit model trained for 30,000 updates on the same units; for the continuous model, a random initialisation reached a lower loss on the development set after 2,000 updates than a start from that transformer (0.83 against 0.95), so we kept the random start. At test time the length comes from the same regression, 64 Euler steps integrate the predicted velocity from Gaussian noise, the result is scaled back, and every frame is replaced by the unit of its nearest centre; the units then go through the same vocoder, ASR model and scoring as the discrete route. On the text of Table 3 (text BLEU 31.46 on the full test set), the continuous model reaches 22.86 ASR-BLEU, 5.24 below the 28.10 of the discrete model (Table 5; $p = 0.001$, paired bootstrap with 1,000 resamples). Both routes produce unit sequences of the same length and their brevity penalties are close (0.874 and 0.866), so the loss lies in the n-gram precisions: 56.0, 32.9, 20.3 and 12.5 against 60.9, 38.9, 26.1 and 17.9 for 1- to 4-grams. This is a single training run decoded with one setting; other numbers of steps, other solvers and other scales of the initial noise were not tried.

Table 5: Generating continuous speech features instead of discrete units in the text-to-unit stage: ASR-BLEU on the text of the LLaDA system of Table 1 (text BLEU 31.46), CVSS-C French–English test set (14,759 utterances). Both models receive the same text and the same unit-sequence lengths, and their units go through the same vocoder and ASR model. The first row is the stage used throughout the paper.

Generated by the text-to-unit model	ASR-BLEU	BP	n-gram precisions ($n = 1$–4)
Discrete units	28.10	0.866	60.9 / 38.9 / 26.1 / 17.9
Continuous features, then nearest centre	22.86	0.874	56.0 / 32.9 / 20.3 / 12.5

F LLaDA, FlexMDM and DreamOn: training details

This appendix gives the training objectives of the LLaDA system of Section 3.2 and of its extensions FlexMDM and DreamOn, with the details of the extensions that Section 3.3 leaves out; Appendix G lists their hyper-parameters.

F.1 LLaDA

For each training example b a time t_b is drawn uniformly from $[0, 1]$ and turned into a masking ratio $p_b = (1 - \epsilon)\, t_b + \epsilon$ with $\epsilon = 10^{-3}$; every slot of the response canvas is masked independently with probability p_b, and the model predicts each masked slot from the speech prefix and the partially masked canvas. The loss is the cross-entropy on the masked slots, EOS pads included, weighted by the inverse of the masking ratio and normalised by the canvas length $L = 64$ and the batch size B:

$$\mathcal{L}_{\mathrm{LLaDA}} = -\frac{1}{BL} \sum_{b=1}^{B} \frac{1}{p_b} \sum_{i=1}^{L} m_{b,i} \log q_\theta(y_{b,i} \mid s_b, \tilde{y}_b)\,, \tag{2}$$

where $y_{b,i}$ is the token in slot i of example b, $m_{b,i} \in \{0, 1\}$ marks the masked slots, $\tilde{y}_b$ is the canvas with those slots replaced by the mask token, s_b is the speech prefix and q_θ is the model's distribution over LLaDA's vocabulary. This is the loss of LLaDA's supervised fine-tuning code (Nie et al., 2025), whose answer length equals the canvas length here because EOS pads count as answer tokens.

F.2 FlexMDM

Model. In the retrofit of Section 3.3.1, a time embedding enters through adaptive layer normalisation, with one zero-initialised scale-and-shift per group of eight blocks; the insertion head reads the final hidden states; and new LoRA adapters of rank 128 are attached to the query, key and value projections of every block and to the vocabulary output head. LLaDA's weights, with the rank-32 adapter of the 9,900-update model folded in, stay frozen; the new adapters, the time embedding, the scale-and-shift layers and the insertion head are newly initialised and train together with the speech encoder and the projector, which start from their values in that model. A sequence of n tokens has $n + 1$ gaps, and one position is appended to the input so that the last gap of a full sequence also has an output.

Corruption and time sampling. Besides the schedules and the weight floor given below, three changes to the paper's recipe are ours. First, the unmasking time is $\beta^{-1}(s + (1 - s)u)$ with $u \sim U[0, 1]$, whereas in the paper its density is $\dot{\beta}(t)/(1 - \beta(s))$ on $[s, 1]$ (Kim et al., 2025, Eq. 6); the two agree for a linear β, but with our $\beta(t) = t^3$ a token inserted at s cannot be unmasked before $s^{1/3}$. Second, t_b is drawn stratified over the batch on $[\epsilon, 1]$ with $\epsilon = 10^{-3}$, as by the authors, but with probability 0.5 it is replaced by a uniform draw on $[\epsilon, 0.4]$, so that early times are seen more often. Third, with probability 0.25 the insertion times of an example are sorted along the position, so that the removed tokens form a suffix.

Insertion head and schedules. The insertion head applies layer normalisation, a hidden layer with GELU and a softplus output scaled by 64 to the final hidden state of each position. Its loss for a gap with true count g and prediction $\hat{g}$ is $\phi(g, \hat{g}) = \hat{g} - g + g(\log g - \log \hat{g})$, with $0 \log 0 = 0$: the negative log-likelihood of g under a Poisson distribution with mean $\hat{g}$, up to a term that depends only on g.

The paper uses linear schedules, $a(t) = \beta(t) = t$, and the code the FlexMDM authors released for their LLaDA retrofit uses $a(t) = e^{-10^{\,1-3t}}$ with a linear β. As motivated in Section 3.3.1, we favour early insertion and delay token commitments: $a(t) = 0.85\, e^{-10^{\,1-6t}} + 0.15\, t$ inserts most tokens early, $a(0.4) \approx 0.88$, while its linear part keeps some insertions arriving late, so that the insertion head also sees incomplete sequences at late times, and $\beta(t) = t^3$ delays commits.

Loss. One update minimises the sum of FlexMDM's unmasking and insertion terms at the sampled times, under our corruption and time sampling, normalised by the batch size B and the capacity L:

$$\mathcal{L}_{\text{FlexMDM}} = \frac{1}{BL} \sum_{b=1}^{B} \left[-w_\beta(t_b) \sum_{i \in M_b} \log q_\theta(y_{b,i} \mid s_b, \tilde{y}_b, t_b) + w_a(t_b) \sum_{i=0}^{n_b} \phi(g_{b,i}, \hat{g}_{b,i}) \right], \quad (3)$$

where $\tilde{y}_b$ is the packed corrupted sequence of example b, with n_b tokens and masked slots M_b, $y_{b,i}$ the clean token behind slot i, $g_{b,i}$ and $\hat{g}_{b,i}$ the true and predicted counts of gap i, s_b the speech prefix and q_θ the unmasking head. The weights are the rate factors of the schedules, $w_\beta(t) = \dot{\beta}(t)/(1-\beta(t))$ and $w_a(t) = \max\{\dot{a}(t)/(1-a(t)),\, 2\}$; the floor of 2 is ours and changes the weight only for $t_b < 0.09$.

F.3 DreamOn

Model. LoRA adapters are attached as for FlexMDM. As there, LLaDA's weights, with the rank-32 adapter of the 9,900-update model folded in, stay frozen; the new adapters are newly initialised and train together with the speech encoder and the projector, which start from their values in that model.

Merging probability. As in the paper, which mixes a static merging probability of 0.5 and a dynamic inverse schedule in equal parts (Wu et al., 2026, Secs. 3.4 and 4.1), the merging probability of an example is 0.5 with probability one half and otherwise $0.5\,(1 - m_b/(n_b + e_b))$, where m_b is the number of masked transcript tokens, so that heavily masked examples merge less.

Loss. The loss is the cross-entropy on the masked positions of the shortened canvas, each weighted by $1 - r_b$; the weighted losses at all positions whose target is an appended delete state are averaged together over the batch and count as a single position, and the sum is divided by the number of the other masked positions in the batch plus one:

$$\mathcal{L}_{\text{DreamOn}} = -\frac{1}{N+1} \sum_{b=1}^{B} (1 - r_b) \left[\sum_{i \in M_b \setminus D_b} \log q_\theta(y_{b,i} \mid s_b, \tilde{y}_b) + \frac{1}{|D|} \sum_{i \in D_b} \log q_\theta(y_{b,i} \mid s_b, \tilde{y}_b) \right], \quad (4)$$

where $\tilde{y}_b$ is the shortened canvas of example b with masked positions M_b, $D_b \subseteq M_b$ holds the positions of its appended delete states, and $N = \sum_b |M_b \setminus D_b|$ and $|D| = \sum_b |D_b|$ are counted over the batch of B examples. The target $y_{b,i}$ is the delete state on D_b and a transcript token or the expand state elsewhere; s_b is the speech prefix and q_θ is the model's distribution over LLaDA's vocabulary. When the batch holds no delete state, the delete term is zero; this guard is ours. Apart from this guard, this is the loss of the training script the authors released, which also weights by one minus the masking ratio and takes the two counts over all examples of a forward pass; the paper takes the counts per sequence and multiplies the weights by the number of masks in the sequence (Wu et al., 2026, Eqs. 2 and 3).

Count head. The count head of Section 3.3.2 is a multilayer perceptron with two hidden layers that reads the mean and the maximum over time of the stacked encoder states and the logarithm of their number. It is fitted with the pinball loss to the 0.25, 0.35 and 0.5 quantiles of the transcript length in LLaDA tokens, and decoding uses its 0.25 quantile.

G HYPER-PARAMETERS

Tables 6 and 7 list the hyper-parameters of the four text-writing systems built on LLaDA, one column per system; those of the Branching Flows system are in Appendix I.

Table 6: Hyper-parameters of the text-writing systems built on LLaDA (part 1 of 2): model, optimisation and the sampled time t_b. The adapter output is scaled by α/r. Batches are limited by padded source filterbank frames (10 ms each); an epoch is one pass over the 207,364 training utterances, and the 9,900-update budget of LLaDA ends 294 batches before its third epoch completes. The first column is the starting checkpoint of the continued model and of the extensions; the other columns count only their additional training. FlexMDM's new modules are the time embedding, the adaptive layer normalisation modulations and the insertion head, trained in full. $\epsilon = 10^{-3}$ throughout.

	LLaDA	LLaDA continued	FlexMDM	DreamOn
Starting point	LLaDA-8B-Base, random projector, released encoder	LLaDA with its adapter folded in	LLaDA with its adapter folded in; fresh time embedding, modulations and insertion head	LLaDA with its adapter folded in
LoRA rank r / α / dropout	32 / 64 / 0	128 / 128 / 0.1	128 / 128 / 0.1	128 / 128 / 0.1
LoRA targets	q, k, v, attention output of every block	q, k, v of every block; vocabulary projection	same	same
Trainable: projector	21.0M	21.0M	21.0M	21.0M
Trainable: adapter	33.6M	117.4M	117.4M	117.4M
Trainable: encoder	33.5M	33.5M	33.5M	33.5M
Trainable: new modules	–	–	285.3M	–
Optimiser	AdamW, betas 0.9 / 0.999, no weight decay	same	same; weight decay 0.1 on the new modules	AdamW, betas 0.9 / 0.999, no weight decay
Peak learning rate, projector and adapter / encoder	1e-4 / 1e-5	same	same; new modules 1e-4	1e-4 / 1e-5
Schedule	linear warm-up 100 updates, cosine decay to a tenth of the peak	warm-up 300 updates, same decay	warm-up 100 updates, same decay	warm-up 300 updates, same decay
Updates (epochs)	9,900 (2.91)	3,970 additional (1.00)	3,970 additional (1.00)	3,970 additional (1.00)
Batch: max. padded source frames per update	28,000	24,000	24,000	24,000
Gradient clipping	1.0	1.0	1.0	1.0
Time t_b	$U[0, 1]$	same	stratified on $[\epsilon, 1]$; with probability 1/2 replaced by $U[\epsilon, 0.4]$	–
GPUs / seed	1 H200 / 42	1 H200 / 42	1 H200 / 42	1 H200 / 42

Table 7: Hyper-parameters of the text-writing systems built on LLaDA (part 2 of 2): masking, the settings specific to FlexMDM and DreamOn, and decoding. DreamOn's n_b and m_b are the transcript length and its number of masked tokens (Section 3.3.2 and Appendix F). NFE counts the calls of the text decoder; the text-to-unit stage adds its own calls.

	LLaDA	LLaDA continued	FlexMDM	DreamOn
Masking ratio	$p_b = (1-\epsilon)\,t_b + \epsilon$, $\epsilon = 10^{-3}$	same	set by the schedules at t_b	$r_b \sim U[0,1]$ per example on the transcript tokens; appended delete states always masked
Insertion schedule $a(t)$ / unmasking schedule $\beta(t)$	–	–	$0.85\,e^{-10^{\,1-6t}} + 0.15\,t$ / t^3 (paper: t / t)	–
Insertion weight floor / suffix corruption probability	–	–	2 / 0.25	–
DreamOn: appended delete states e_b	–	–	–	uniform on $\{0, \ldots, \min(7, 64-n_b)-1\}$, none if $n_b = 64$ (paper: 0 to 64)
DreamOn: merging probability	–	–	–	0.5 or $0.5\,(1 - m_b/(n_b+e_b))$, each with probability 1/2 (paper: 0.5, static and dynamic inverse 1:1)
Canvas	64 slots	64 slots	capacity 64, grown from empty	64 positions, starting from the count head's length
NFE (text decoder)	–	64	64	variable, at most 192
FlexMDM decoding: unmask count / insertion	–	–	Poisson draw, seed 0, margin confidence / credit (paper: Poisson draw / Poisson draws)	–
DreamOn decoding: commits per call / ranking / token	–	–	–	1, all remaining on a 192nd call / top probability / most likely (paper: entropy ranking, temperature 0.2, top-p 0.9)
DreamOn count head: inputs / loss / quantile used	–	–	–	mean, max and log length of the stacked encoder states / pinball at 0.25, 0.35, 0.5 / 0.25, rounded, clipped to [1, 64]

H BRANCHING FLOWS: TRAINING DETAILS

Initialisation. The blocks and the time embedding of the decoder of Section 3.4 start from random or zero initialisation and its output layer starts at zero, with one exception, the text-piece embedding table, which is copied from the first-pass text decoder of the UnitY model after 30,000 updates of a speech-to-unit training run of ours on the same data, in which the UnitY front end, that text decoder included, was fine-tuned with an added text cross-entropy term.

Split and deletion heads. Two linear heads read the output of the last decoder block, which the output layer also reads, and the weights and biases of both start at zero. The split head gives $\hat{\mu} = \exp(\mathrm{clip}(z_\mathrm{s}, -100, 11))$, where z_s is the head's output and clip limits it to $[-100, 11]$. Its loss against the element's number n of further splits in the training example is $\phi(n, \hat{\mu}) = \hat{\mu} - n \log \hat{\mu} - n + n \log n$, with $0 \log 0 = 0$, which equals the negative log-likelihood of n under a Poisson distribution with mean $\hat{\mu}$ up to a term that depends on n only, and is zero at $\hat{\mu} = n$. The deletion head gives $\hat{\rho}$ as the sigmoid of its output; its loss is the binary cross-entropy against the element's deletion flag d. With the zero start, every element begins the second stage with $\hat{\mu} = 1$ and $\hat{\rho} = 1/2$. These are the split and deletion terms of the paper's objective (Billera et al., 2025, Eq. 1); the shift of the split term by $n \log n - n$ follows the authors' code.

Training examples. For a transcript of l pieces, the number of copies is drawn from a Poisson distribution with mean $0.2\,l$ (Billera et al., 2025, App. F.1.2); with a one-element start this is the deletion rate $d_r = 1.2$ of Billera et al. (2025, App. A.2). Each copy repeats a piece of the transcript, drawn uniformly with replacement from the original pieces, and is inserted directly before or after it, with probability one half each. With probability one half, and only if the original is not yet marked, the original is then marked for deletion; otherwise the copy is marked, as in the authors' code. The copies of a piece therefore sit next to it, and each copy adds exactly one deletion mark. The pieces and copies, in order, are the leaves of a binary tree built by repeatedly merging a uniformly chosen pair of adjacent nodes until a single root remains (Billera et al., 2025, App. A.3). Split times are drawn from the root down: a node with W leaves splits into its two children at a time drawn after its parent's split time, or after 0 for the root, with hazard rate $(W-1)\,h_\mathrm{s}(t)$, where h_s is the hazard rate of Beta(1,2); each marked leaf is removed at a time drawn uniformly between its parent's split time and 1. The training time t is drawn uniformly between 0 and 1 and, with probability one half, replaced by a uniformly chosen split time of the tree when the tree has any, as in the authors' released example. At a training time t the canvas holds the elements present at t: an element stands for one leaf or, when its split has not yet happened at time t, for a set of leaves, and removed leaves are absent; the anchor of an element is its text piece in the first case and the mask token in the second (Billera et al., 2025, App. A.4), and its token state is obtained by running the discrete base process down the tree from a single mask token at time 0. The base process uses the schedules $\kappa_1(t) = F(t)$ and $\kappa_2(t) = \omega_u\,(1 - F(t))\,F(t)$ of Billera et al. (2025, App. D.2), where $F(t) = 3t^2 - 2t^3$ is the distribution function of Beta(2,2) and $\omega_u = 0.2$: started from the mask token at time 0, with a fixed anchor, its state at time t is the anchor with probability $\kappa_1(t)$, a symbol drawn uniformly from all 6,001 output symbols with probability $\kappa_2(t)$, and otherwise still the mask token. These schedules, the single-element start and the number of copies follow the paper's setting for small molecules (Billera et al., 2025, App. F.1.2); the Beta(1,2) split-time distribution follows its setting for antibody sequences (Billera et al., 2025, App. F.3.2), and the uniform deletion time is used in both. On the branch that leads to a node, from its parent's split time t_0, or 0 for the root, to its own split time, the token state follows the base process of Billera et al. (2025, App. D.2) towards the node's anchor: from its state at t_0, the state at time t is the anchor with probability $\kappa_1(t) - \kappa_1(t_0)\,\kappa_3(t)/\kappa_3(t_0)$, a symbol drawn uniformly from all 6,001 output symbols, the mask token included as in the authors' code, with probability $\kappa_2(t) - \kappa_2(t_0)\,\kappa_3(t)/\kappa_3(t_0)$, and unchanged with probability $\kappa_3(t)/\kappa_3(t_0)$, where $\kappa_3 = 1 - \kappa_1 - \kappa_2$. The root starts as the mask token at time 0, and an element present at time t holds the state of its branch at t. The targets of an element are its anchor, its number of further splits, one less than its number of leaves, marked leaves included, and a deletion flag that is one only for an element that is a single marked leaf.

Loss. The first-stage loss is

$$\mathcal{L}_1 = \frac{1}{|M|} \sum_{i \in M} \ell_{0.01}\big(q_\theta(\cdot \mid i), y_i\big) + \lambda\,\mathcal{L}_\mathrm{aux}, \tag{5}$$

where M holds the masked non-padding positions of the batch, y_i is the text piece at position i, $q_\theta(\cdot \mid i)$ is the decoder's distribution over the text vocabulary at i, $\ell_{0.01}$ is the cross-entropy with label smoothing 0.01, whose form in each stage is given below, $\mathcal{L}_{\text{aux}}$ is the auxiliary decoder's cross-entropy averaged over its non-padding target tokens, and $\lambda = 1$. The second stage averages three Branching Flows terms over the canvas elements of the batch, each weighted by $w(t) = 1/(1.2-t)$:

$$\mathcal{L}_{\text{tok}} = \frac{1}{|E|} \sum_{e \in E} w(t_e)\, \ell_{0.01}\big(q_\theta(\cdot \mid e), a_e\big),$$

$$\mathcal{L}_{\text{split}} = \frac{1}{|E|} \sum_{e \in E} w(t_e)\, \phi(n_e, \hat{\mu}_e),$$

$$\mathcal{L}_{\text{del}} = \frac{1}{|E|} \sum_{e \in E} w(t_e)\, \text{BCE}(\hat{\rho}_e, d_e),$$

and the second-stage loss is

$$\mathcal{L}_2 = \mathcal{L}_{\text{tok}} + \mathcal{L}_{\text{split}} + \mathcal{L}_{\text{del}} + \lambda\, \mathcal{L}_{\text{aux}}, \tag{6}$$

where E holds the canvas elements of the batch, t_e is the training time of the example of element e, a_e, n_e and d_e are its anchor, number of further splits and deletion flag, $q_\theta(\cdot \mid e)$ is the decoder's distribution over the 6,001 output symbols at e, ϕ and BCE are the head losses above, and $\lambda = 1$ as in the first stage. The token term is taken at every element, and the anchor of an element that stands for several leaves is the mask token, which is why the output layer needs the mask row. The weight w is the one of the authors' released example; the label smoothing and the auxiliary term are ours.

Label smoothing. With $\varepsilon = 0.01$, the first stage uses $\ell_{0.01}(q, y) = -(1-\varepsilon)\log q(y) - \frac{\varepsilon}{V-1}\sum_{v \neq y} \log q(v)$ over the $V = 6{,}000$ text pieces, and the second stage uses $\ell_{0.01}(q, a) = -(1-\varepsilon)\log q(a) - \frac{\varepsilon}{K}\sum_{k=1}^{K} \log q(k)$ over all $K = 6{,}001$ output symbols, the target and the mask token included.

I HYPER-PARAMETERS OF THE BRANCHING FLOWS SYSTEM

Tables 8 and 9 list the hyper-parameters of the Branching Flows system of Section 3.4, one column per training stage.

Table 8: Hyper-parameters of the Branching Flows system of Section 3.4 (part 1 of 2): model, optimisation and losses, one column per training stage. Batches are limited by padded source filterbank frames, 160,000 per GPU on each of the two GPUs; an epoch is 296 updates. All values in this part are our choices.

	Stage 1	Stage 2
Starting point	released UnitY encoder and first-pass text decoder (auxiliary); new decoder with the piece embeddings of our speech-to-unit run	stage-1 checkpoint; new mask row of the output layer and two new heads, all zero
Trainable: decoder / encoder / auxiliary decoder	32.3M / 33.5M / 18.8M	same
Optimiser	Adam (fairseq implementation), betas 0.9 / 0.98, eps 1e-8, decoupled weight decay 0.01	same, fresh optimiser state
Peak learning rate	1e-3	1e-3
Schedule	linear warm-up 10,000 updates, then inverse square root	linear warm-up 500 updates, then inverse square root
Updates (epochs)	50,000 (168.9), the checkpoint with the highest development BLEU on a fixed canvas of the target length among those every 5,000 updates from 30,000 to 60,000	17,760 (60), the last checkpoint
Batch: max. padded source frames per update	320,000	320,000
Gradient clipping	1.0	1.0
Dropout / attention / activation	0.3 / 0.3 / 0.3	same
SpecAugment in training	yes	yes
Max. target length	64 pieces	same
Label smoothing: decoder / auxiliary	0.01 / 0	same
Auxiliary loss weight	1	same
GPUs / seed	2 H200 / 42	2 H200 / 42

Table 9: Hyper-parameters of the Branching Flows system (part 2 of 2): training time, Branching Flows process and decoding, one column per training stage; the rows from the start state down to the canvas cap apply to stage 2 and its decoding. $\epsilon = 10^{-3}$. F is the distribution function of Beta(2,2). In parentheses, the source of a value taken from Branching Flows: F.1.2 and F.3.2 are the paper's settings for molecules and for antibody sequences (Billera et al., 2025, App. F.1.2, F.3.2), and "example" is the authors' released example; all other values are our choices.

	Stage 1	Stage 2
Time t	$U[\epsilon, 1-\epsilon)$; each piece masked with probability $1-t$	$U(0,1)$; with probability 1/2 replaced by a split time of the tree (example)
Start state x_0	–	one masked element (F.1.2)
Base-process schedules κ_1 / κ_2	–	F / $\omega_u\,(1-F)\,F$, $\omega_u = 0.2$ (F.1.2)
Split-time / deletion-time distribution	–	Beta(1,2) (F.3.2, example) / $U(0,1)$ (F.1.2, F.3.2)
Deletion copies	–	Poisson, mean 0.2 times the transcript length; drawn with replacement, either side of the original (F.1.2)
Loss weight	–	$1/(1.2-t)$ (example)
NFE (text decoder)	–	64
Decoding: re-masking threshold / split multiplier m	–	0.5 / 1.2
Decoding: canvas cap	–	4,000 elements, empty output beyond