Block3D: Efficient Text-to-3D Generation via Block-Wise Diffusion Supplementary Material

Anonymous submission

This document supplements the method, data, evaluation, qualitative results, and release materials in the main paper. Sections A–E follow the section identifiers cited by the main paper and provide the complete algorithms, experimental protocol, full qualitative prompts, and code/data-package index.

A Complete Method Details

A.1 Frozen Shape Representation and Generator

Shape representation. Block3D retains the frozen vector-quantized shape representation of Cube (Foundation AI Team et al. 2025). A mesh S is encoded as a fixed-length sequence

$$\begin{aligned} x &= (x_1, \ldots, x_N), \qquad N = 1024, \\ x_i &\in \{0, \ldots, V-1\}. \end{aligned} \tag{A.1}$$

with codebook size $V = 16384$. The Cube shape encoder is used to prepare shape-code supervision before generator training. The corresponding shape decoder D_{shape} remains frozen and converts only a completed sequence of valid shape codes into the output mesh. The auxiliary mask symbol $[M]$ reuses the inherited padding-token identifier for input embedding lookup, but it is excluded from the V-way output normalization and is never passed to the shape decoder.

Condition representation. Given a text prompt, the frozen CLIP ViT-L/14 encoder (Radford et al. 2021) produces 77 token-level condition features. Cube also supports an optional projected bounding-box token. All experiments reported in the main paper use text alone, so no bounding-box token is appended in the reported training or inference runs. Classifier-free condition dropout replaces the retained text condition with the frozen empty-text encoding; if bounding-box conditioning is enabled in another setting, its unconditional counterpart is a zero bounding box.

Trainable generator. The conditional shape prior is initialized from Cube's 23-layer DualStream RoFormer generator (Foundation AI Team et al. 2025), with 12 attention heads and hidden width 1536. Block3D uses the dual-stream path and bypasses the inherited single-stream layer. The first V input token embeddings are initialized from a learned projection of the frozen VQ codebook. During fine-tuning, the VQ shape encoder, VQ codebook, CLIP text encoder, and shape decoder remain frozen; only the text-to-shape generator is updated. At inference, all components are frozen.

A.2 Training Attention and Position IDs

Block notation. For block size B, the N shape positions are divided into $K = \lceil N/B \rceil$ contiguous blocks. Block r contains

$$I_r = \{(r-1)B + 1, \ldots, \min(rB, N)\}. \tag{A.2}$$

Training concatenates a clean sequence x and a corrupted sequence $\tilde{x}$, while keeping a single condition sequence C. We write $x^{(r)} = x_{I_r}$ and $\tilde{x}^{(r)} = \tilde{x}_{I_r}$.

Complete visibility rule. The attention mask adapts the vectorized clean/noisy construction of Block Diffusion (Arriola et al. 2025) to fixed-length Cube codes. Let $A(q, k) = 1$ indicate that query q can attend to key k. Condition queries are isolated from the shape sequence:

$$A(C, C) = 1, \qquad A(C, x^{(s)}) = A(C, \tilde{x}^{(s)}) = 0. \tag{A.3}$$

Every shape query can attend to all condition tokens. A clean query in block r attends block-bidirectionally to its own clean block and to every preceding clean block, but never to the corrupted copy:

$$\begin{aligned} A(x^{(r)}, C) &= 1, \\ A(x^{(r)}, x^{(s)}) &= \mathbf{1}[s \le r], \\ A(x^{(r)}, \tilde{x}^{(s)}) &= 0. \end{aligned} \tag{A.4}$$

A corrupted query in block r attends to clean blocks strictly before r and bidirectionally to the corrupted tokens in its own block:

$$\begin{aligned} A(\tilde{x}^{(r)}, C) &= 1, \\ A(\tilde{x}^{(r)}, x^{(s)}) &= \mathbf{1}[s < r], \\ A(\tilde{x}^{(r)}, \tilde{x}^{(s)}) &= \mathbf{1}[s = r]. \end{aligned} \tag{A.5}$$

Consequently, a corrupted query cannot access the clean target tokens of its own block, any future clean block, or a corrupted state from another block. All corrupted block contexts can nevertheless be evaluated in one forward pass because each block receives its own clean teacher-forced prefix.

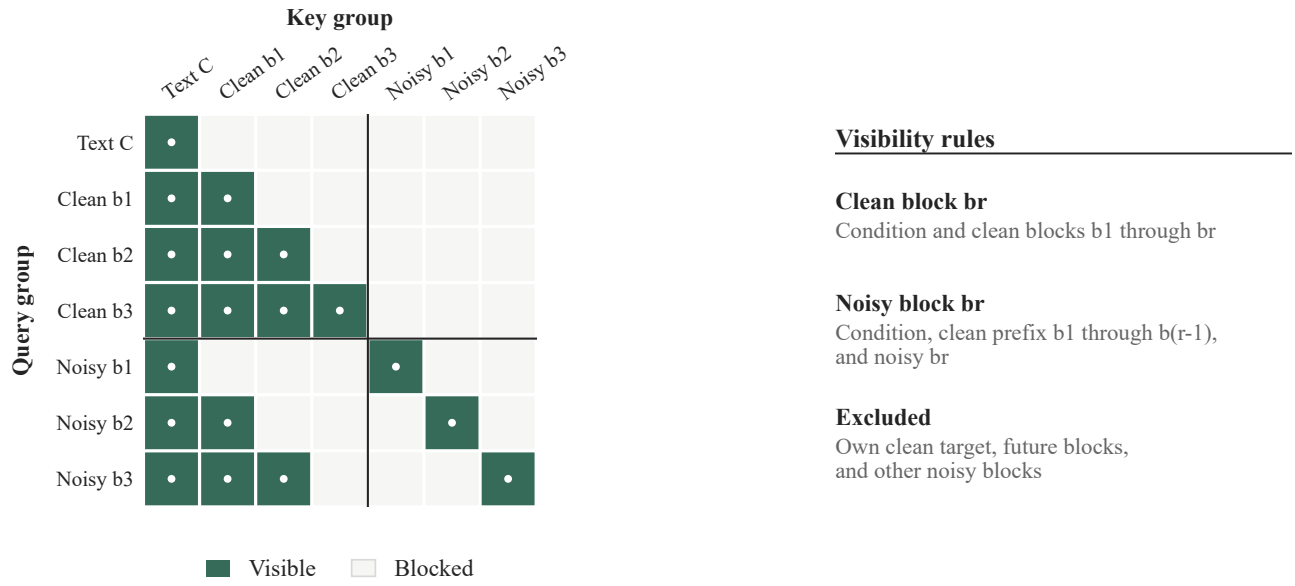

Figure A: Block-level visualization of the training attention mask. Green cells denote visible query–key group pairs. A corrupted query reads the condition, its clean teacher-forced prefix, and the corrupted copy of its own block; it cannot read the clean target of that block, a future block, or another corrupted block. The diagram expands each block to a single group for readability; the corresponding token-level rules are given in Equations A.3–A.5.

Logical position IDs. The clean and corrupted copies share logical shape positions:

$$p(x_i) = p(\tilde{x}_i) = i, \qquad i = 1, \ldots, N. \tag{A.6}$$

Thus duplication for training changes the physical sequence layout but not the RoFormer position assigned to a shape code. At inference, a committed prefix position $\hat{x}_i$ retains its global ID i, and an active-block state $z_{k,i}$ also uses global ID i. Prefix queries use token-causal visibility, whereas active-block queries attend to the complete prefix and bidirectionally within the active block. Future blocks are absent from the inference sequence.

Training–inference distinction. The training construction supplies block-bidirectional, teacher-forced clean prefixes, while inference conditions on token-causal, model-generated prefixes. Prefix caching reduces repeated computation but does not remove this exposure difference. Completed blocks are immutable in both the cache and the generated sequence.

Figure A makes the leakage boundary explicit. Bidirectional visibility is confined to the matching block, while the clean prefix remains strictly earlier than the corrupted target block. The duplicated clean and corrupted streams therefore vectorize supervision over all block indices without exposing any block to its own target codes.

A.3 Corruption, Rollout, and Residual Loss

Sample-level corruption stream. For each training sample, a single variable $a \sim \mathrm{Bernoulli}(\rho)$ chooses either mask-to-token (M2T, $a = 0$) or token-to-token (T2T, $a = 1$) corruption for the complete shape sequence. For every block k, a corruption rate $\tau_k \sim \mathcal{U}(t_{\min}, t_{\max})$ is drawn independently, followed by $e_i \sim \mathrm{Bernoulli}(\tau_k)$ for $i \in I_k$. The initial state is

$$\tilde{x}_i^{(0)} = \begin{cases} [M], & e_i = 1,\ a = 0, \\ u_i, & e_i = 1,\ a = 1, \\ x_i, & e_i = 0, \end{cases} \tag{A.7}$$

where u_i is sampled uniformly from the $V - 1$ codes different from x_i. The reported setting uses $(t_{\min}, t_{\max}) = (0.45, 0.95)$ and $\rho = 0.5$. The sampled corruption rate changes only the input state and is not provided to the generator as a time embedding.

Condition dropout. Classifier-free condition dropout (Ho and Salimans 2022) is sampled once per batch element with probability 0.1. The realized retained or dropped condition is reused for both the rollout forward pass and the gradient-carrying loss forward pass, preventing the two passes from receiving inconsistent conditions.

One-step model rollout. Block3D uses one model-based rollout ($R = 1$), inspired by the multi-turn-forward augmentation of LLaDA2.1 (Bie et al. 2026). Starting from $z_k^{(0)} = \tilde{x}_{I_k}^{(0)}$, an unguided, no-gradient forward pass evaluates all corrupted blocks with the attention mask in Section A.2. Each block then applies the M2T and T2T update rules from Section A.4 using the first reveal quota q_0. Under $B = 64$ and $T = 4$, $q_0 = 16$. All selected positions are updated simultaneously to obtain $z_k^{(1)}$, and these per-block states are reassembled into $\tilde{x}^{(R)}$. The rollout output may contain masks, correct codes, and model-produced incorrect codes regardless of which initial corruption stream was selected.

Residual objective. A separate gradient-carrying forward pass evaluates $[x; \tilde{x}^{(R)}]$. Let $\pi^{(R)}_{\theta,n,i}$ be the softmax over the first V logits for batch item n and position i, and define the residual set

$$\mathcal{D} = \{(n,i) : \tilde{x}^{(R)}_{n,i} \neq x_{n,i}\}. \tag{A.8}$$

The training loss is

$$\mathcal{L} = -\frac{1}{|\mathcal{D}| + \epsilon} \sum_{(n,i)\in\mathcal{D}} \log \pi^{(R)}_{\theta,n,i}(x_{n,i}), \qquad \epsilon = 10^{-8}. \tag{A.9}$$

Reduction is performed over all residual tokens in the batch rather than averaging separate per-sample losses. A sample with no residual positions contributes no term to the numerator. If the entire batch has $\mathcal{D} = \emptyset$, the numerator is empty and the batch loss is zero. Padding or otherwise invalid shape positions are excluded before $\mathcal{D}$ is formed.

Algorithm A.1 Edit-aware training state construction

Require: Clean codes x; condition C; $B, T, t_{\min}, t_{\max}, \rho, \eta_M, \eta_T$

Sample condition dropout once for each batch item
Sample one stream variable a for each batch item
for each block k **do**
  Sample τ_k and construct $\tilde{x}^{(0)}_{I_k}$ by Equation A.7
end for
Run one no-gradient, unguided forward pass on $[x; \tilde{x}^{(0)}]$
Apply one simultaneous M2T/T2T update independently in every block
Assemble the updated blocks into $\tilde{x}^{(R)}$
Run a separate gradient-carrying forward pass on $[x; \tilde{x}^{(R)}]$
Form $\mathcal{D}$ and compute Equation A.9
return Residual denoising loss $\mathcal{L}$

A.4 Complete Inference Algorithm

Guided proposal and conditional confidence. For active block k, let $z^{(0)}_{k,i} = [M]$ for $i \in I_k$ and $n_k = |I_k|$. At iteration $s \in \{0, \ldots, T-1\}$, the conditional and unconditional branches produce ℓ^+_s and ℓ^-_s. Classifier-free guidance gives

$$\ell^g_s = (1+\gamma_s)\ell^+_s - \gamma_s \ell^-_s, \qquad \gamma_s = g\frac{T-s}{T}. \tag{A.10}$$

When $g = 0$, the unconditional branch is omitted. The guided logits propose a candidate, while the conditional logits score its acceptance:

$$\begin{aligned} \hat{z}_{s,i} &= \arg\max_{0\le v<V} \ell^g_{s,i,v}, \\ \alpha_{s,i} &= \mathrm{softmax}(\ell^+_{s,i,0:V-1})[\hat{z}_{s,i}]. \end{aligned} \tag{A.11}$$

An argmax tie is resolved by choosing the smallest shape-code index.

Reveal quota and update sets. Iteration s receives the deterministic minimum reveal quota

$$q_s = \left\lfloor \frac{n_k}{T} \right\rfloor + \mathbf{1}[s < n_k \bmod T], \qquad \sum_{s=0}^{T-1} q_s = n_k. \tag{A.12}$$

Let $\mathcal{M}_s = \{i \in I_k : z^{(s)}_{k,i} = [M]\}$. The two update sets adapted from LLaDA2.1 (Bie et al. 2026) are

$$\begin{aligned} \mathcal{U}^{\mathrm{M2T}}_s = \{&i \in \mathcal{M}_s : \alpha_{s,i} > \eta_M\} \\ &\cup \ \mathrm{TopK}_{i\in\mathcal{M}_s}\big(\alpha_{s,i}, \min(q_s, |\mathcal{M}_s|)\big), \end{aligned} \tag{A.13}$$

$$\mathcal{U}^{\mathrm{T2T}}_s = \{i \in I_k \setminus \mathcal{M}_s : \hat{z}_{s,i} \neq z^{(s)}_{k,i},\ \alpha_{s,i} > \eta_T\}. \tag{A.14}$$

For TopK confidence ties, the smaller global shape position is selected first. TopK returns the empty set when its requested size is zero. The union in Equation A.13 means that confidence-qualified M2T updates may exceed q_s; the quota is a lower bound on reveal progress, not an upper bound on the number of updates.

Simultaneous update and completion. All accepted positions are updated from the same iteration state:

$$z^{(s+1)}_{k,i} = \begin{cases} \hat{z}_{s,i}, & i \in \mathcal{U}^{\mathrm{M2T}}_s \cup \mathcal{U}^{\mathrm{T2T}}_s, \\ z^{(s)}_{k,i}, & \text{otherwise.} \end{cases} \tag{A.15}$$

If $m_s = |\mathcal{M}_s|$, the fallback term reveals at least $\min(q_s, m_s)$ positions. Therefore

$$m_{s+1} \le \max(m_s - q_s, 0). \tag{A.16}$$

Together with $m_0 = n_k$ and $\sum_s q_s = n_k$, Equation A.16 gives $m_T = 0$. Every block is consequently complete after at most T iterations. For a short block with $n_k < T$, the first n_k quotas equal one and the remaining quotas equal zero; the same argument applies without padding the block to size B.

Algorithm A.2 Complete bounded editable block sampler

Require: Condition C^+; optional C^-; $N, V, B, T, g, \eta_M, \eta_T$

Set $K \leftarrow \lceil N/B \rceil$ and initialize $\hat{x} \leftarrow [M]^N$
for $k = 1$ to K **do**
  Set $z^{(0)}_k \leftarrow [M]^{n_k}$ and $S_k \leftarrow T$
  Build the batched condition/prefix cache for block k
  for $s = 0$ to $T-1$ **do**
    Evaluate ℓ^+_s and, if $g > 0$, ℓ^-_s
    Compute $\hat{z}_s$ and α_s by Equations A.10–A.11
    Form $\mathcal{M}_s$, q_s, $\mathcal{U}^{\mathrm{M2T}}_s$, and $\mathcal{U}^{\mathrm{T2T}}_s$
    if $\mathcal{M}_s = \emptyset$ and both update sets are empty **then**
      Set $S_k \leftarrow s$; **break**
    end if
    Update all active positions simultaneously by Equation A.15
  end for
  Commit $\hat{x}_{I_k} \leftarrow z^{(S_k)}_k$ and freeze block k
end for
return $D_{\mathrm{shape}}(\hat{x})$

The reported sampler uses deterministic argmax decoding, without top-p sampling, with $B = 64$, $T = 4$, $(\eta_M, \eta_T) = (0.95, 0.9)$, and $g = 3.0$. Because $N = 1024$, this setting has $K = 16$ full blocks and no short final block. The per-iteration guidance coefficients are $(3.0, 2.25, 1.5, 0.75)$.

A.5 Prefix Cache and Complexity

Cache construction. At the start of block k, the condition and committed prefix $\hat{x}^{(<k)}$ are encoded for the conditional branch and, when $g > 0$, for the unconditional branch. The two branches are concatenated along the batch dimension. Their condition and prefix key/value states remain unchanged across the T active-block updates and are therefore reused. Only the active-block states are recomputed after an M2T/T2T update. Once block k is committed, its codes become part of the immutable prefix and the cache is rebuilt for block $k + 1$.

Model-call accounting. For K blocks and at most T updates per block, decoding uses K cache-building calls and at most KT active-block calls. Conditional and unconditional branches are executed in the same batched call, so the batched-call upper bound is

$$K + KT = K(T + 1). \tag{A.17}$$

When $g > 0$, each batched call contains two logical branch evaluations, giving at most

$$2K(T + 1) \tag{A.18}$$

logical evaluations. With $N = 1024$, $B = 64$, and $T = 4$, the upper bounds are 80 batched calls and 160 logical branch evaluations. Early stopping can reduce active-block calls but does not increase either bound. This accounting measures generator invocations; it does not include the final frozen shape-decoder call.

B Data and Training Reproducibility

B.1 Data Split and Prompt Provenance

The source pool is formed from TRELLIS-500K records (Xiang et al. 2025), each of which associates a 3D asset with a paired text description. The paired text is used as the text-to-shape condition. Each evaluation object retains its paired prompt, and the same stored prompt is provided to every generation method without method-specific editing.

Using seed 42, we sample 100 valid asset records for evaluation and assign the stable indices 0000–0099. The recovered manifest contains 100 distinct prompts and 100 distinct source-mesh paths: 53 records from Objaverse-XL Sketchfab, 46 from Objaverse-XL GitHub, and one from ABO, all distributed through TRELLIS-500K. Every evaluated method receives this same ordered prompt list and produces one output for each prompt. Deterministic decoders use their fixed argmax path, while stochastic external baselines use seed 42.

Before fine-tuning, each evaluation record is matched against the candidate pool by its normalized pair-record path and, when that key is unavailable, by its normalized source-mesh path. Split construction stops if an evaluation record cannot be matched or if the number of excluded source records differs from 100. This enforces exact asset-level disjointness between the evaluation list and the fine-tuning pool; it does not claim category-level or semantic near-duplicate removal. The accompanying Code and Data Package includes `random100_evaluation_manifest.jsonl`, which records the stable index, fixed prompt, source collection, normalized TRELLIS-500K mesh path, pair-record key, and bounding-box vector for every evaluation item. The package identifies rather than redistributes the third-party target meshes; they are resolved from the referenced TRELLIS-500K snapshot.

To characterize the fixed list without introducing a hand-assigned category taxonomy, we report properties that can be recomputed directly from the ordered manifest. For target bounding-box vector $b_j = (b_{j,x}, b_{j,y}, b_{j,z})$, define the scale-invariant anisotropy

$$a_j = \frac{\max_d b_{j,d}}{\min_d b_{j,d}}. \tag{B.1}$$

Prompt length ranges from 10 to 38 whitespace-delimited words, with median 19.0. The median a_j is 2.15, and 36 of 100 targets have $a_j > 3$, so the paired set includes a substantial fraction of elongated or flattened shapes rather than only near-isotropic objects. These bounding-box vectors are used only by the common evaluator; no evaluated generator receives them as input.

B.2 Data Preprocessing

We apply the released Cube preprocessing (Foundation AI Team et al. 2025) before tokenization. Meshes are triangulated, non-finite geometry and zero-area faces are removed, and unreferenced vertices are discarded. All remaining nonempty connected components are retained. The original orientation is preserved: no category-specific rotation, canonical-pose fitting, or ICP alignment is applied. Each valid mesh is centered at its axis-aligned bounding-box midpoint and uniformly rescaled by its longest box side to fit the normalized Cube coordinate domain.

Every preprocessed training mesh and paired evaluation target is converted into a length-1024 discrete shape-code sequence with the same frozen Cube tokenizer. The sequence is cached together with its TRELLIS identifier and provides the clean target x used by the corruption process in Section A.3. An input is excluded only when preprocessing leaves no finite triangle surface. At inference, the frozen shape decoder receives a sequence only after all 1024 positions contain valid codebook entries. This single tokenizer keeps the representation fixed across Block3D training, the controlled Cube baseline, and paired targets; no learned geometry component is updated during Block3D fine-tuning.

B.3 Training Configuration

We initialize from the released Cube checkpoint and optimize only the text-to-shape generator on 300K training objects. Training runs for 35K optimizer updates in bfloat16 on four NVIDIA A100 80GB GPUs with an effective global batch size of 40 and seed 42. AdamW (Loshchilov and Hutter 2019) uses a learning rate of 10^{-4}, $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, and zero weight decay. The learning rate increases linearly from zero to 10^{-4} during the first 100 updates and remains constant thereafter, and gradients are clipped to global norm 1.0 before each optimizer update. Classifier-free condition dropout is applied with probability 0.1.

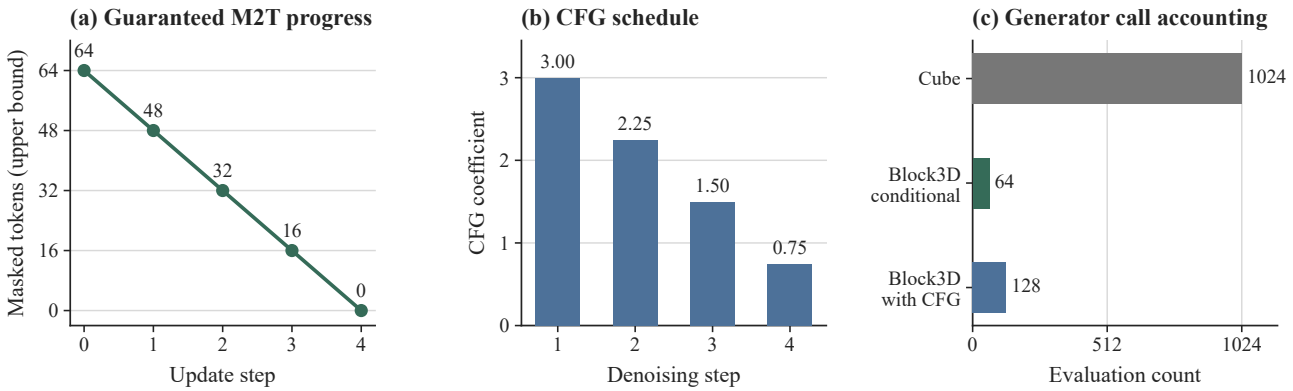

Figure B: Analytical diagnostics for the main configuration. Left: the deterministic quota reveals at least 16 positions per step, so the number of masks is bounded by $64 \to 48 \to 32 \to 16 \to 0$. Middle: Equation A.10 yields guidance coefficients $(3.00, 2.25, 1.50, 0.75)$. Right: excluding the K cache-building calls, Block3D performs at most $KT = 64$ conditional active-block evaluations, or $2KT = 128$ logical branches with CFG, whereas token-wise Cube performs 1,024 sequential token evaluations. These counts describe the generator schedule and are not substitutes for measured end-to-end latency.

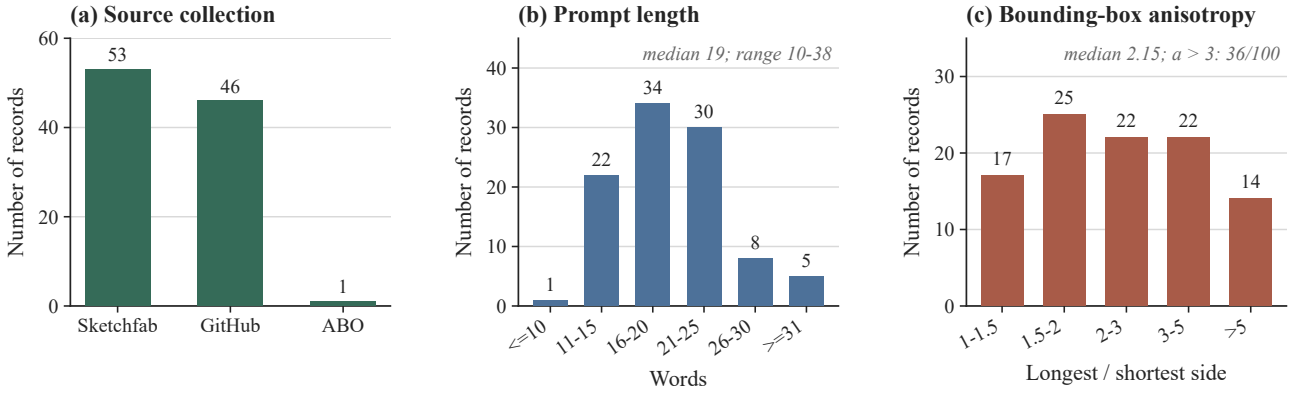

Figure C: Recomputable profile of the fixed random100 evaluation manifest. The source counts sum to 100; prompt-length bins use the stored text verbatim; and anisotropy uses Equation B.1. The distribution reflects the seeded TRELLIS-500K sample and is not manually balanced by source, prompt length, or shape extent.

The training corruption range is $[0.45, 0.95]$, the T2T stream is selected with probability 0.5, and the main model uses one no-gradient model-based rollout. The VQ tokenizer, codebook, CLIP ViT-L/14 text encoder, and shape decoder remain frozen, and the 35K-step checkpoint is used for evaluation. All block-size, denoising-step, and M2T-only variants start from the same released Cube checkpoint and are independently trained for the same 35K-update budget.

The update and batch settings determine the amount of training data presented to the optimizer. Let $S = 35{,}000$ be the number of updates, $G = 4$ the number of workers, $b = 10$ the local batch size, and $M = 300{,}000$ the number of training records. Then

$$M_{\text{seen}} = SGb = 1.4 \times 10^6, \qquad \frac{M_{\text{seen}}}{M} \approx 4.67. \quad \text{(B.2)}$$

Here M_{seen} counts sample presentations, not unique objects; shuffling is renewed by the distributed sampler. For the default full block, let $\mathcal{D}_k^{(0)}$ contain the initially corrupted positions and let n_{T2T} count T2T-stream samples in one global batch. The initial corruption process gives

$$\begin{aligned} \mathbb{E}[\tau_k] &= \frac{0.45 + 0.95}{2} = 0.70, \\ \mathbb{E}[|\mathcal{D}_k^{(0)}|] &= 64(0.70) = 44.8, \\ \mathbb{E}[n_{\text{T2T}}] &= 0.5(40) = 20. \end{aligned} \quad \text{(B.3)}$$

Thus an average initial block retains roughly 19.2 clean anchors before the model-based rollout, while an expected 20 samples in each global batch begin from the T2T corruption stream. These are analytical expectations under the sampled corruption process, not additional empirical measurements.

The reference software environment is Linux with Python 3.10 and PyTorch 2.2.2 or later built for CUDA. Training uses four NVIDIA A100 80GB GPUs, and the latency protocol in Section C.3 uses one A100 80GB GPU on the same host for every method. The host CPU performs data loading, mesh import, and process orchestration, but CPU throughput is not a reported comparison variable. The released trainer records the operating-system platform, CPU model

and logical core count, host memory, Python and PyTorch versions, CUDA runtime, GPU names, and GPU memory in `training_run.json` for each reproduced run.

Optimization and checkpointing. Gradient accumulation is one, so every loaded global batch produces one optimizer update. Under bfloat16 autocast, no float16 loss scaler is active. Gradients are checked for a finite global norm, clipped to 1.0, and then passed to AdamW; the optimizer is cleared with `set_to_none=True` before the next update. EMA is disabled. The main configuration performs no periodic validation, sample generation, or geometry evaluation during optimization, preventing an auxiliary evaluation loop from selecting the reported checkpoint. Model-only checkpoints are written every 5K updates, and the final 35K checkpoint is the one used by the reported evaluation.

B.4 Parameter Selection

The reported operating point is $B = 64$, $T = 4$, $(\eta_M, \eta_T) = (0.95, 0.9)$, guidance coefficient $g = 3.0$, corruption range $[0.45, 0.95]$, T2T stream probability 0.5, and one rollout step. Block size controls the number of left-to-right stages, while T controls the fixed number of active-block refinement opportunities.

The block-size study trains $B \in \{32, 64, 96, 128, 256\}$ with $T = 4$. Increasing B reduces latency but eventually weakens paired geometric fidelity; $B = 64$ is used as the reported quality–latency balance. The denoising-step study fixes $B = 64$ and trains $T \in \{4, 8, 12, 20\}$. Additional steps increase measured latency without providing monotonic gains in the reported geometry metrics, so the main configuration uses $T = 4$. The T2T study independently trains an M2T-only variant and the complete M2T+T2T model under the same optimization budget.

The remaining values are fixed globally rather than adjusted per prompt. The M2T/T2T thresholds $(0.95, 0.9)$ follow the conservative public decoding defaults of LLaDA2.1 (Bie et al. 2026); the guidance coefficient $g = 3.0$ is applied through the within-block decay in Equation A.10. The corruption interval $[0.45, 0.95]$ exposes the model to substantially incomplete blocks while retaining clean anchors in most training states. One rollout is the lowest-cost nontrivial model-based augmentation and introduces model-generated errors without multiplying the gradient-carrying training pass. These settings are shared by the block-size and denoising-step ablations unless the corresponding variable is under study; the M2T-only comparison uses its dedicated M2T training configuration while retaining the same initialization and optimization budget.

C Complete Evaluation Protocol

C.1 Geometry Metrics

Generated and target meshes are loaded through the same geometry routine. Non-finite values, degenerate and duplicate faces, and unreferenced vertices are removed. Each mesh is centered at its own axis-aligned bounding-box midpoint and uniformly rescaled so that its longest box side equals 1.92, the Cube evaluation domain. This transform preserves aspect ratio and orientation; no rotational alignment, ICP, or prompt-specific registration is applied. From each transformed mesh, 8,192 points are sampled with probability proportional to triangle area; the evaluator uses base seed 0 and adds the stable record index, so record j uses NumPy seed j for its paired surface draws. Each point receives the unit face normal of its sampled triangle. Let $P = \{(p_i, n_i)\}$ and $Q = \{(q_j, m_j)\}$ denote the generated and target samples. The unsquared symmetric Chamfer distance is

$$\begin{aligned}\mathrm{CD\text{-}L1}(P, Q) = &\frac{1}{2|P|} \sum_{p \in P} \min_{q \in Q} \|p - q\|_2 \\ &+ \frac{1}{2|Q|} \sum_{q \in Q} \min_{p \in P} \|q - p\|_2.\end{aligned} \tag{C.1}$$

For each point, let $\nu_Q(p)$ be its nearest sampled target point and $\nu_P(q)$ the nearest generated point. Normal consistency is the symmetric mean of the absolute normal dot products:

$$\begin{aligned}\mathrm{NC}(P, Q) = &\frac{1}{2|P|} \sum_{p \in P} |n_p^\top n_{\nu_Q(p)}| \\ &+ \frac{1}{2|Q|} \sum_{q \in Q} |m_q^\top m_{\nu_P(q)}|.\end{aligned} \tag{C.2}$$

Let b_Q be the target bounding-box vector stored with the paired TRELLIS-500K record after Cube normalization, and let $d_Q = \|b_Q\|_2$. All 100 reported records contain this vector; the evaluator falls back to the transformed target extent only when it is absent. With $\delta = 0.01 d_Q$, precision is the fraction of generated samples within δ of the target, recall is the fraction of target samples within δ of the generation, and

$$F@1\% = \frac{2 \,\mathrm{precision}\, \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}. \tag{C.3}$$

The main paper reports CD-L1, NC, and F@1% to three decimals. Nearest neighbors are computed in Euclidean distance on the 8,192-point sets, and the same neighbors are used for the corresponding normal terms. These definitions follow standard point-sampled geometry evaluation (Fan, Su, and Guibas 2017; Mescheder et al. 2019).

The implementation samples positive-area triangles proportionally to area and draws uniform barycentric coordinates with the square-root transform. Record j initializes one NumPy generator with seed j; target and generated surfaces are drawn sequentially from it. Two SciPy `cKDTree` queries compute both nearest-neighbor directions. The evaluator uses unsquared distances, reuses the same indices for normals, and saves per-sample values and status fields before macro-averaging.

C.2 CLIPScore

Text–shape alignment is evaluated from eight fixed views using CLIP ViT-L/14 (Radford et al. 2021). Each mesh is centered and isotropically normalized by its axis-aligned bounding box, assigned the same neutral-gray material, and rendered at 512×512 pixels with PyTorch3D from eight azimuths separated by $45°$. Camera elevation, distance, field of view, background, and lighting are fixed and shared by

all methods. For prompt feature t and image feature v_r from view r, the score is

$$\mathrm{CLIPScore}(t, S) = \frac{100}{8} \sum_{r=1}^{8} \max\left(\frac{t^\top v_r}{\|t\|_2 \|v_r\|_2}, 0 \right). \tag{C.4}$$

The official CLIP preprocessing center-crops each view and applies the released image normalization. Prompts use the same 77-token truncation and text preprocessing as the frozen condition encoder. Equation C.4 is the positive cosine form of CLIPScore (Hessel et al. 2021); scores are averaged over views and then over the 100 prompts and reported to two decimals.

C.3 Latency

Latency is measured in inference mode with batch size one on one NVIDIA A100 80GB GPU. Each model remains resident on the device, untimed warm-up runs are completed before measurement, and CUDA is synchronized immediately before and after every timed generation. The same ordered set of 100 TRELLIS-500K prompts is used for every method, with one timed generation per prompt. End-to-end time includes condition encoding, shape-code generation, and mesh decoding; model loading, checkpoint transfer, visualization, and disk I/O are excluded. Mean, median, P90, and standard deviation are computed over the 100 recorded generations.

Specifically, let $t_{(1)} \le \cdots \le t_{(100)}$ be the sorted per-prompt times and let $\bar{t}$ denote their mean. The recorded summaries use

$$\bar{t} = \frac{1}{100} \sum_{j=1}^{100} t_j, \qquad \mathrm{median}(t) = \frac{t_{(50)} + t_{(51)}}{2},$$
$$\sigma_t = \sqrt{\frac{1}{100} \sum_{j=1}^{100} (t_j - \bar{t})^2}, \qquad \mathrm{P90}(t) = t_{(90)}. \tag{C.5}$$

The standard deviation is therefore the population standard deviation over the fixed evaluation list, and P90 uses the nearest-rank definition. Timing records remain paired with stable sample IDs so aggregate values can be traced back to individual prompts.

Block3D and Cube run in bfloat16 and share the same frozen Cube geometry components. Block3D uses $B = 64$, $T = 4$, $g = 3.0$, and deterministic argmax decoding. The external methods retain their official inference precision, sampling schedule, kernels, and native mesh decoders. All mesh-decoding stages execute on the timed device and are included in the reported end-to-end latency. This measured protocol is separate from the analytical generator-call accounting in Section A.5.

C.4 Baseline Settings

Controlled baseline. Cube (Foundation AI Team et al. 2025) is the controlled baseline because it shares the released initialization, shape tokenizer, codebook, CLIP text encoder, shape decoder, 300K training subset, and 35K-step optimization budget with Block3D. The generator objective and decoding schedule differ. Block3D uses the block-causal editable decoder described in Section A.4, with $B = 64$, $T = 4$, and $g = 3.0$.

External baselines. ShapeLLM-Omni (Ye et al. 2025), TRELLIS-text (Xiang et al. 2025), and AR3D-R1 (Tang et al. 2025) are external text-to-3D baselines. They use their official inference pipelines. Their native representations, sampling schedules, and postprocessing are retained rather than forced into the Cube pipeline.

For TRELLIS-text, we use the released `TRELLIS-text-xlarge` pipeline with 25 sparse-structure steps, 25 structured-latent steps, guidance scale 7.5 for both stages, seed 42, and its native mesh decoder. ShapeLLM-Omni and AR3D-R1 use their released text-to-3D checkpoints, default decoding parameters, and native mesh extraction. Cube uses greedy autoregressive decoding of all 1,024 codes. Block3D uses the deterministic sampler in Algorithm A.2 for the quality evaluation. No method receives the target mesh, a bounding box, or method-specific prompt editing, and no method-specific geometry cleanup is applied before the common evaluation transform.

C.5 Statistical Scope and Evaluation Boundaries

The evaluation contains 100 paired prompts. Each method generates one shape per prompt under the same single-generation budget. Geometry and CLIP values are macro-averaged over the 100 ordered outputs, so each prompt contributes equal weight; latency is summarized over the same 100 prompts by its mean, median, P90, and standard deviation. Because every method is evaluated on the same prompt–target pairs, per-prompt comparisons are paired even though the main tables report aggregate values.

CD-L1, NC, and F@1% quantify fidelity to the paired reference, while eight-view CLIPScore measures reference-free prompt alignment. An output is valid when it contains finite vertices, at least one nondegenerate face, and a nonzero bounding-box extent after native mesh extraction. Generation, geometry evaluation, and CLIP evaluation succeeded for all 100 records in every row reported in the main quality table; therefore no missing-value imputation or failure penalty affects the reported means. The evaluator records generation success, mesh validity, and metric status for each sample and does not regenerate a failed output.

D Qualitative Results and Prompts

This section documents all qualitative figures before listing their exact prompts. Figures D and E separate the teaser presentation from its generated components, Figure F restores the complete prompts used in the main comparison, and Figure G broadens the visual coverage with additional Block3D outputs.

D.1 Teaser Prompts and Isolated Assets

The teaser in the main paper is a Blender scene assembled from Block3D mesh outputs. It is not a direct multi-object or scene-generation output. Generated mesh objects were imported into Blender (Blender Online Community 2018),

Figure D: Four views of the manually assembled teaser arena. The layout, repeated placement of objects, display colors, lighting, and background are Blender presentation choices; Block3D generates the component meshes rather than the complete arena composition.

Figure E: Isolated mesh-object inventory for the teaser scene. Labels 000–041 match the media manifest and provide a stable visual index for the individual turntable files.

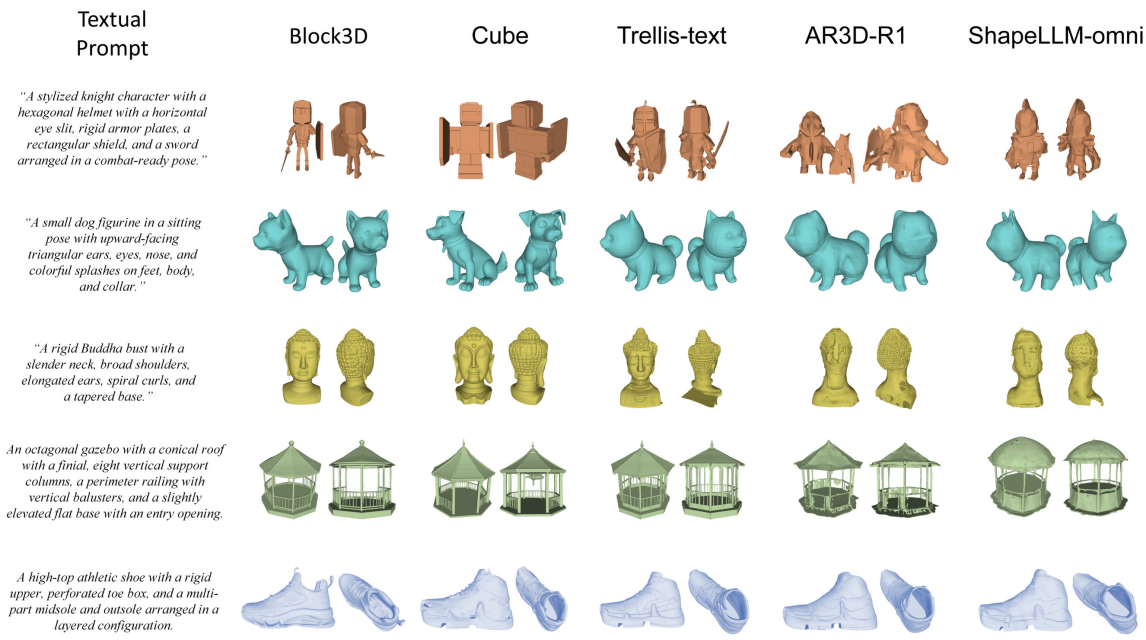

Figure F: Full-prompt version of the qualitative comparison in the main paper. Columns show Block3D, Cube, TRELLIS-text, AR3D-R1, and ShapeLLM-Omni; each result is rendered from front and back views.

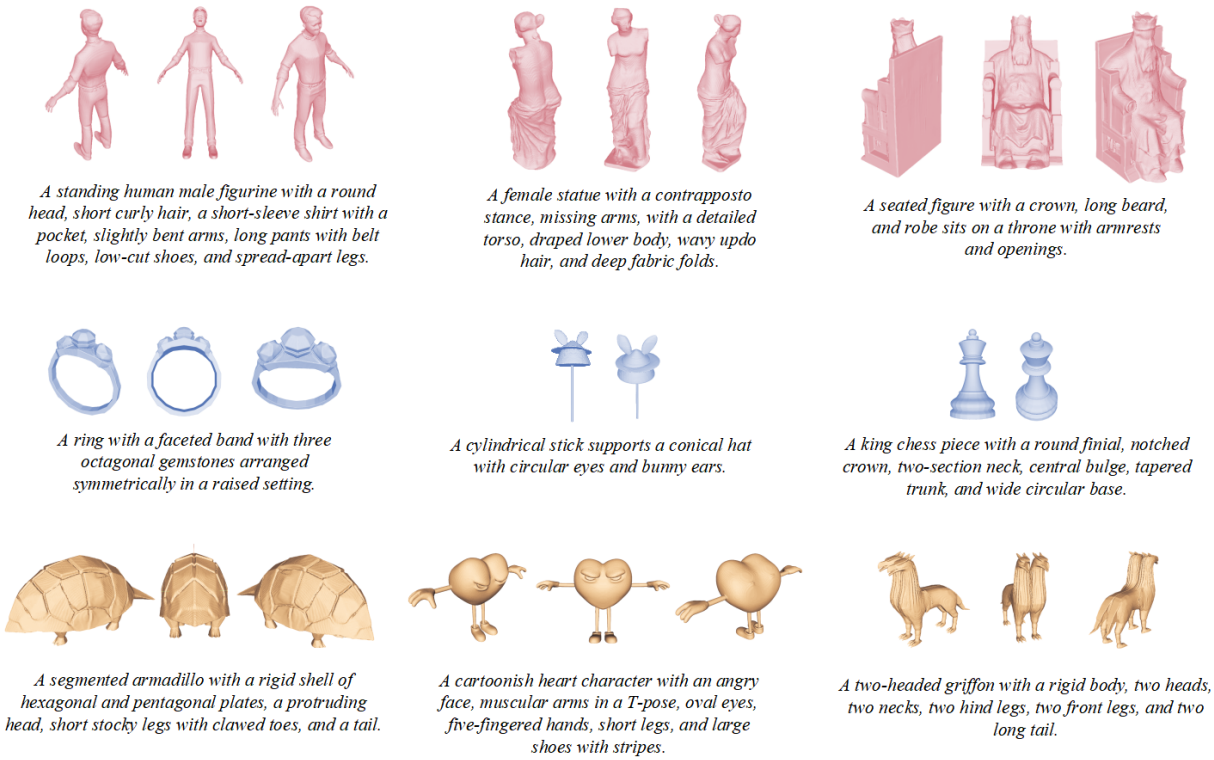

Figure G: Additional Block3D text-to-shape results. Each group contains multiple viewpoints of a single generated mesh, with its complete text prompt shown directly below.

arranged into the arena, assigned display materials, and rendered with a shared scene, camera, and lighting setup.

Each unique component is generated once with the deterministic Block3D configuration in Section A.4. Blender duplicates, transforms, colors, and arranges the meshes but does not modify their geometry.

The assembled arena therefore demonstrates the composability of independently generated assets. It does not claim that Block3D generates a complete scene layout or a jointly conditioned multi-object scene.

Component prompts. The identifiers below match Figure E. Each prompt generates one object. Ranges denote repeated scene instances that reuse the same generated mesh; multiplicity and placement are determined only during scene assembly.

- **000 (knight).** "A stylized armored knight with a helmet, plate armor, shield, and sword."
- **001 (dog).** "A small dog companion with upright ears."
- **002/014 (throne).** "A high-backed wooden throne with a stepped base."
- **003–023 (fence).** "A short wooden palisade fence with pointed stakes."
- **024/036 (stairs).** "A broad three-step stone stair."
- **025/026 (torch).** "A medieval standing torch on a narrow pole."
- **027 (rack).** "A simple rack holding upright spears."
- **028 (barrels).** "Two wooden barrels placed side by side."
- **029–031 (barrier).** "Two short posts connected by a heavy chain."
- **032 (marker).** "A circular stone floor marker."
- **033 (tree).** "A stylized conical pine tree."
- **034/035 (banner).** "A rectangular hanging arena banner."
- **037 (stump).** "A cut tree stump with visible roots."
- **038 (rocks).** "A clustered pile of rocks."
- **039 (chest).** "A reinforced wooden storage chest."
- **040 (hammock).** "A cloth hammock suspended between two supports."
- **041 (spears).** "A tied bundle of long spears."

D.2 Figure 4 Prompts and Results

The complete prompts corresponding to the five rows of Figure 4 in the main paper are:

1. "A stylized knight character with a hexagonal helmet with a horizontal eye slit, rigid armor plates, a rectangular shield, and a sword arranged in a combat-ready pose."
2. "A small dog figurine in a sitting pose with upward-facing triangular ears, eyes, nose, and colorful splashes on feet, body, and collar."
3. "A rigid Buddha bust with a slender neck, broad shoulders, elongated ears, spiral curls, and a tapered base."
4. "An octagonal gazebo with a conical roof with a finial, eight vertical support columns, a perimeter railing with vertical balusters, and a slightly elevated flat base with an entry opening."
5. "A high-top athletic shoe with a rigid upper, perforated toe box, and a multi-part midsole and outsole arranged in a layered configuration."

Figure F reproduces the comparison without truncating these prompts. Each method is shown from the same two semantic viewpoints used in the main figure. The red boxes in the main paper identify missing or distorted local geometry and are presentation annotations rather than additional measurements.

These five prompts constitute a separate qualitative panel chosen to cover an articulated character, an animal, a bust, architecture, and a manufactured object. They are not members of the random100 paired evaluation set and do not contribute to the quantitative averages. The prompt list is fixed and shared across methods, and front and back views use the same camera and normalization for all methods.

D.3 Additional Text-to-Shape Examples

Figure G presents nine additional Block3D outputs produced with the deterministic configuration in Section A.4. Each group shows two or three rendered viewpoints of one generated mesh, rather than separate generations, and prints the complete generation prompt below the corresponding result. The examples extend the qualitative coverage to human figures and statues, accessories and manufactured objects, and stylized animals and characters. This panel is illustrative and is not included in any reported metric or model-selection decision. Its complementary viewpoints expose the same decoded geometry, including thin structures, bilateral forms, and articulated silhouettes that may be hidden from a single view.

E Code and Data Package

E.1 Archive Contents and Implementation Map

The anonymous source archive is organized around three reproducibility paths. The `block3d/training/` package contains the data loader, clean/corrupted attention construction, sample-level M2T/T2T corruption, no-gradient rollout, residual objective, and distributed trainer. The `block3d/inference/` package contains condition preparation for the reported text-only path, the bounded editable sampler, CFG schedule, and prefix-cache execution. Shared masks, reveal schedules, and M2T/T2T update sets are implemented in `block3d/model/gpt/block_diffusion_utils.py`; geometry and eight-view CLIPScore are implemented in `block3d/evaluation.py`. The archive also includes paper-aligned YAML configurations, command-line scripts, `pyproject.toml`, a README with complete commands, the upstream-compatible license, and the ordered `random100_evaluation_manifest.jsonl`.

The archive contains source and small metadata only. It does not redistribute the Cube (Foundation AI Team et al. 2025) checkpoint, frozen shape tokenizer, CLIP ViT-L/14 (Radford et al. 2021) weights, TRELLIS-500K (Xiang et al. 2025) assets, external-baseline repositories, generated meshes, or trained Block3D checkpoints. The README identifies the required Cube release as

`Roblox/cube3d-v0.5`, specifies the expected local artifact layout, and distinguishes the released initialization from random initialization, which is not the protocol reported in the paper.

E.2 Training and Inference Entry Points

The main experiment is specified by `block3d/configs/block3d.yaml` and `block3d/configs/train_block3d.yaml`; the independently trained M2T-only comparison has a corresponding configuration pair. The script `scripts/materialize_ablation_configs.py` writes the independently trained configurations for $B \in \{32, 96, 128, 256\}$ and $T \in \{8, 12, 20\}$, while the main files provide $B = 64, T = 4$. Every training configuration retains the 35K-step optimization budget, global batch size 40, seed 42, frozen geometry components, and text-only condition. Distributed training is launched through `torchrun` and `block3d.train_block_diffusion`; the final 35K-step checkpoint is evaluated without validation-based model selection.

Text-to-mesh inference uses `python -m block3d.generate`. For the reported protocol, the command omits both the optional bounding-box argument and top-p, disables optional PyMeshLab postprocessing, uses $B = 64, T = 4, g = 3.0$, and writes one native decoded mesh for one prompt. The release deliberately excludes cluster-specific batch scheduling: the ordered evaluation manifest provides the 100 prompts and stable sample IDs, and the same single-prompt entry point is invoked once for each record. This matches the one-generation-per-prompt scope in Section C.5.

E.3 Data, Evaluation, and Environment Records

The split script `scripts/prepare_block_diffusion_eval_split.py` matches the fixed 100-record manifest to the TRELLIS-500K (Xiang et al. 2025) source pool, stops on any missing record or overlap, excludes exactly those 100 assets, and selects the 300K training records with seed 42. It writes the training manifest, the fixed evaluation manifest, the excluded-record list, and a split summary. Third-party mesh targets are referenced by source-relative paths and pair-record keys rather than redistributed.

The script `scripts/evaluate_generation.py` consumes per-method `samples.jsonl` files and writes per-sample JSONL/CSV records plus an aggregate `summary.json`. Its paper command uses 8,192 surface samples, base seed 0 with the stable record-index offset described in Section C.1, F@1% from the stored target bounding-box diagonal, and eight PyTorch3D renders for CLIPScore. Invalid or missing outputs are recorded rather than regenerated. Latency values use the boundary in Section C.3; `block3d/benchmarking.py` computes the reported mean, median, P90, and population standard deviation from the ordered records.

Package installation is defined by `pyproject.toml` for Python 3.10 and PyTorch 2.2.2 or later with CUDA. PyTorch3D is required for the paper-aligned CLIP renderer; Blender and PyMeshLab are optional mesh-import/postprocessing dependencies and do not change the text-only generator. Each training launch stores its resolved model/training configuration, command, random seed, git revision, platform, CPU and memory metadata, Python/PyTorch/CUDA versions, and GPU inventory. These records keep machine-dependent reproduction details separate from the fixed paper protocol stated in Sections B and C.

References

Arriola, M.; Gokaslan, A.; Chiu, J. T.; Yang, Z.; Qi, Z.; Han, J.; Sahoo, S. S.; and Kuleshov, V. 2025. Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models. In *ICLR*.

Bie, T.; Cao, M.; Cao, X.; Chen, B.; Chen, F.; Chen, K.; Du, L.; Feng, D.; Feng, H.; Gong, M.; et al. 2026. LLaDA2.1: Speeding Up Text Diffusion via Token Editing. arXiv:2602.08676.

Blender Online Community. 2018. Blender: A 3D Modelling and Rendering Package. https://www.blender.org. Accessed: 2026-07-29.

Fan, H.; Su, H.; and Guibas, L. J. 2017. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. In *CVPR*, 2463–2471.

Foundation AI Team; Bhat, K.; Khanna, N.; Channa, K.; Zhou, T.; Zhu, Y.; Sun, X.; Shang, C.; Sudarshan, A.; et al. 2025. Cube: A Roblox View of 3D Intelligence. arXiv:2503.15475.

Hessel, J.; Holtzman, A.; Forbes, M.; Le Bras, R.; and Choi, Y. 2021. CLIPScore: A Reference-Free Evaluation Metric for Image Captioning. In *EMNLP*, 7514–7528. Association for Computational Linguistics.

Ho, J.; and Salimans, T. 2022. Classifier-Free Diffusion Guidance. arXiv:2207.12598.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *ICLR*.

Mescheder, L.; Oechsle, M.; Niemeyer, M.; Nowozin, S.; and Geiger, A. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *CVPR*, 4460–4470.

Radford, A.; Kim, J. W.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; Krueger, G.; and Sutskever, I. 2021. Learning Transferable Visual Models from Natural Language Supervision. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, 8748–8763. PMLR.

Tang, Y.; Guo, Z.; Zhu, K.; Zhang, R.; Chen, Q.; Jiang, D.; Liu, J.; Zeng, B.; Song, H.; Qu, D.; Bai, T.; Xu, D.; Zhang, W.; and Zhao, B. 2025. Are We Ready for RL in Text-to-3D Generation? A Progressive Investigation. arXiv:2512.10949.

Xiang, J.; Lv, Z.; Xu, S.; Deng, Y.; Wang, R.; Zhang, B.; Chen, D.; Tong, X.; and Yang, J. 2025. Structured 3D Latents for Scalable and Versatile 3D Generation. In *CVPR*, 21469–21480.

Ye, J.; Wang, Z.; Zhao, R.; Xie, S.; and Zhu, J. 2025. ShapeLLM-Omni: A Native Multimodal LLM for 3D Generation and Understanding. arXiv:2506.01853.