SchemaForge: Distilling Ultra-Large Foundation Models into Edge SLMs for Real-Time Enterprise JSON Extraction

A Comparative Study of Gemma-4 Teachers and MiniCPM5-1B

Arjhine A. Ty

August 2026 · Version 1.0

Abstract

Structured entity extraction—converting heterogeneous unstructured business documents into strongly-typed, schema-conformant JSON—is one of the highest-volume workloads in enterprise retrieval-augmented generation (RAG) and transactional automation. The prevailing solution is to deploy a 30B+ parameter foundation model behind a constrained-decoding wrapper. This is operationally untenable at scale: in our measurements a `gemma-4-31B`-class teacher requires ≈38.5 GB of VRAM and sustains only 12.40 tokens/second of greedy decode throughput on a single 96 GB accelerator, leaving room for only two concurrent workers.

We present **SchemaForge**, a sequence-level knowledge-distillation framework that compresses the structural reasoning and JSON formatting strictness of Gemma-4 teachers into `openbmb/MiniCPM5-1B`, a dense 1.08-billion-parameter edge Small Language Model. SchemaForge combines a hard-target cross-entropy objective with a temperature-scaled, log-space soft-target KL divergence under $\mathcal{L}_{KD} = \alpha\mathcal{L}_{CE} + (1-\alpha)\tau^2\mathcal{L}_{KL}$ ($\alpha = 0.5$, $\tau = 2.0$), and resolves the teacher–student vocabulary mismatch ($|\mathcal{V}_T| = 256{,}000 \to |\mathcal{V}_S| = 130{,}560$) via dual-tokenizer cross-encoding with shared-subspace logit projection.

The distilled student attains a **0.0% JSON syntax error rate** and **field-level F1 of 1.000** across a five-domain in-domain enterprise benchmark suite ($n = 5$ documents), against 34.2% error and 0.612 F1 for the undistilled base model. It sustains **61.91–76.27 tokens/second** in a **≈2.4 GB VRAM** footprint—a **16.0**× memory reduction and **5.0**×–**6.2**× throughput improvement over the 31B teacher, permitting **36 concurrent workers** per 96 GB card (≈2,746 tok/s aggregate).

Our central scientific finding is a sharp negative result about prompt fragility. Across three controlled iterations, identical data, loss, and hyperparameters with only a changed instruction prefix moved zero-shot validity on the public `suneeldk/text-json` benchmark from **0.0% to 70.0% and back to 0.0%**. Distillation at this scale transfers a *format-conditioned* skill, not a format-invariant one. We formalise this as the *train–inference template gap* and derive a practical mitigation that we recommend as mandatory for any sub-2B structured-output deployment.

Keywords: knowledge distillation, small language models, structured generation, JSON extraction, prompt sensitivity, edge inference, vLLM, enterprise RAG.

Statement of evaluation scale. The in-domain suite comprises $n = 5$ curated documents and the winning checkpoint was distilled on $n = 5$ training samples. The 1.000 F1 and 0.0% error rate are therefore exact-match results on a small, curated set, not population estimates. Section 10 states this in full and is intended as a load-bearing part of the paper. **This is a v1 release**: a second training and evaluation campaign against a real-world corpus, with an expanded metric set, multi-seed variance, and competitive baselines, is scoped in §11.3. The accuracy figures below should be read as provisional pending that run.

Contents

1 Introduction

1.1 The Structured Extraction Bottleneck

Enterprise document intelligence pipelines are, in aggregate, a text-to-JSON problem. Accounts-payable automation reads invoices into `{invoice_number, vendor_name, invoice_date, subtotal, tax, grand_total}`. Freight settlement reads bills of lading into `{bill_of_lading, carrier_name, ship_date, container_count, freight_cost}`. In every case the input is heterogeneous and adversarially formatted by whichever upstream system emitted it; the output is a rigid, strongly-typed schema that a downstream database, ERP, or RAG index will reject outright if a single brace is unbalanced.

The industry default—point a large instruction-tuned foundation model at the problem—works, and produces three compounding operational failures.

(1) Memory pressure that forecloses multi-tenancy. A 31B-parameter model in `bfloat16` requires roughly 62 GB for weights alone; even under quantization the working set we measured is ≈38.5 GB including KV cache. On a 96 GB RTX PRO 6000 Blackwell this admits only *two* concurrent workers at 90% memory utilisation—against 36 for the distilled student.

(2) Throughput below SLA. We measure 12.40 tokens/second of greedy decode for the 31B teacher. A typical extraction emits 120–250 JSON tokens, implying 10–20 seconds per document. Interactive flows target sub-2-second response; bulk runs of 10^6 documents become physically impossible on any reasonable budget.

(3) Cost per document that does not amortise. Because throughput is low and memory forbids batching, marginal cost is dominated by GPU-seconds at the highest instance tier. There is no batching lever to pull.

Critically, *the capability being paid for is not the capability being used.* A 31B model's value lies in open-domain reasoning, long-horizon planning, and code synthesis. Invoice extraction requires reliable span identification, light numeric normalisation, and absolute fidelity to a JSON

grammar. This mismatch is precisely the setting in which knowledge distillation is theoretically well-motivated.

1.2 Contributions

C1 A reproducible distillation recipe for sub-2B structured extractors: complete objective, hyperparameter bounds, and dataset scaling regime (§4, §6).

C2 A dual-tokenizer cross-encoding and logit-projection scheme permitting KL-based distillation between models with incompatible vocabularies (256,000 → 130,560) without embedding surgery (§4.3).

C3 Four low-level runtime compatibility patches required to train legacy `trust_remote_code` architectures under PyTorch 2.5 / `transformers` 5.x / Python 3.12, including a depth-scaling fix without which the 24-layer student silently produces `NaN` logits (§5).

C4 *The train–inference template gap*: a quantified negative result—three controlled iterations isolating prompt-header formatting, producing a 70-percentage-point accuracy swing (§7).

C5 A complete production deployment protocol: HuggingFace publication, model card specification, vLLM serving, schema-constrained decoding, and capacity planning (§9).

2 Related Work

Knowledge distillation. The foundational formulation is due to Hinton et al. [7], who introduced temperature-softened logit matching with the now-standard τ^2 gradient-rescaling term. Their insight— that the *relative* probabilities a teacher assigns to incorrect classes encode a similarity structure absent from one-hot labels—motivates our soft-target term. Bucilǔa et al. [4] anticipated the model-compression framing. Romero et al. [15] extended supervision to intermediate representations; we deliberately do *not* use hidden-state matching, since teacher and student have no principled layer correspondence.

Sequence-level distillation. Kim and Rush [10] established sequence-level KD for machine translation. SchemaForge is sequence-level in this sense: training targets are teacher-generated JSON completions, not human-annotated gold. Sanh et al. [16] and Jiao et al. [9] demonstrated the approach at scale. MiniLLM [6] and GKD [2] address the exposure-bias mismatch arising when a student is trained on teacher trajectories but evaluated on its own—an unexploited improvement here (§11.2).

Structured and constrained generation. Willard and Louf [18] reformulate constrained decoding as finite-state-machine-guided token masking, achieving provable grammar conformance. This literature is complementary rather than competing: constrained decoding guarantees *syntax* but cannot guarantee *semantics*—a grammar-constrained model will emit a well-formed object with the wrong vendor name. We recommend layering both (§9.4).

Small language models. The MiniCPM line [8] argues that carefully-scaled sub-3B models can match 7B–13B models on targeted benchmarks via depth-scaled residuals and a wide-vocabulary tokenizer; Phi-3 [1] makes the parallel argument from data quality. Gemma [5] provides open-weight teachers at multiple scales.

Prompt sensitivity. Sclar et al. [17] demonstrate performance variation of up to 76 accuracy points under semantically equivalent formatting perturbations; Lu et al. [13] show analogous sensitivity to example ordering; Mizrahi et al. [14] advocate multi-prompt evaluation as standard practice. Our contribution to this thread is specific: distillation into a small student does not merely inherit this brittleness—*it concentrates it*, because limited capacity causes the student to bind learned behaviour tightly to the exact training prefix.

3 Problem Formulation

3.1 Task Definition

Let $x \in \Sigma^*$ denote an unstructured source document and $\mathcal{S}$ a target schema, a finite set of typed field names

$$\mathcal{S} = \{(k_1, \theta_1), \ldots, (k_m, \theta_m)\}, \qquad \theta_i \in \{\texttt{str}, \texttt{int}, \texttt{float}, \texttt{date}, \texttt{array}\}.$$

The extraction task is to learn $f_\Theta : (x, \mathcal{S}) \mapsto y \in \mathcal{J}(\mathcal{S})$, where $\mathcal{J}(\mathcal{S})$ is the set of strings that are (i) syntactically valid JSON under RFC 8259 [3], and (ii) conformant to $\mathcal{S}$. Because f_Θ is an autoregressive language model, generation factorises as

$$P_\Theta(y \mid x, \mathcal{S}) = \prod_{t=1}^{T} P_\Theta(y_t \mid y_{<t}, x, \mathcal{S}),$$

and the schema enters the conditioning *only through the prompt surface form*. This is the structural reason the finding of §7 is possible: the schema is not an architectural constraint, it is a string.

3.2 Evaluation Metrics

(M1) JSON syntax validity rate. Validity $= \frac{1}{N}\sum_i \mathbb{1}[\texttt{json.loads}(\hat{y}_i)$ succeeds$]$, ErrorRate $= 1 -$ Validity. A necessary but wholly insufficient condition: `{}` is valid JSON.

(M2) Field-level extraction F1. Treating each (key, value) pair as a retrievable item against reference $y^\star$,

$$P = \frac{|\hat{K} \cap K^\star|}{|\hat{K}|}, \quad R = \frac{|\hat{K} \cap K^\star|}{|K^\star|}, \quad F_1 = \frac{2PR}{P+R},$$

with type-aware normalisation: numerics coerced to float and compared within $\varepsilon = 10^{-6}$; dates normalised to ISO-8601; strings compared after NFKC normalisation and whitespace collapse. An unparseable generation contributes $F_1 = 0$.

(M3) Decode throughput. Generated tokens per wall-clock second under greedy decoding, batch size 1, excluding model load.

(M4) Peak VRAM footprint. `torch.cuda.max_memory_allocated()` over the full generation, inclusive of weights, activations, and KV cache.

Why all four are required. Each metric is trivially gameable in isolation: emit `{}` always (100% validity, $F_1 \approx 0$); truncate at 8 tokens (excellent throughput, no content); load in 4-bit with no KV cache (minimal VRAM, degraded accuracy). The joint frontier is what matters, and we report all four for every configuration on identical hardware.

Dataset	Role	Size	Provenance
SchemaForge suite (BMK-01–05)	In-domain eval	$n = 5$ docs	Hand-authored, 5 verticals
Iteration-2 set	Training (winner)	$n = 5$ samples	Teacher-generated, canonical template
Iteration-1 set	Training	$n = 20$ samples	Teacher-generated, chat-token template
Iteration-3 set	Training	$n = 15$ samples	Teacher-generated, persona template
`suneeldk/text-json`	Zero-shot eval	2,000 available	Public HF, real documents

Table 1: Datasets. Small training sizes are deliberate—the research question was how little supervision suffices when the signal is a teacher's full logit distribution over a narrow task. They are also the primary limitation of this work (§10).

3.3 Datasets

4 The SchemaForge Distillation Framework

4.1 The Multi-Task Distillation Objective

SchemaForge's objective is the convex combination

$$\boxed{\mathcal{L}_{KD} \;=\; \alpha\,\mathcal{L}_{CE} \;+\; (1-\alpha)\,\tau^2\,\mathcal{L}_{KL}} \tag{1}$$

with $\alpha = 0.5$ and $\tau = 2.0$.

Hard-target term.

$$\mathcal{L}_{CE} = -\sum_{t=1}^{T} \log P_S(y_t \mid y_{<t}, x) = -\sum_{t=1}^{T} \log \frac{\exp\left(z_S^{(y_t)}(t)\right)}{\sum_{v \in \mathcal{V}_S} \exp\left(z_S^{(v)}(t)\right)}. \tag{2}$$

This anchors the student to exact structural tokens. For JSON this is not a formality: the delimiters `{ } [ ] " : ,` are positions where the teacher's distribution is near-degenerate—probability mass ≈ 1 on a single token—and therefore positions where the soft term carries almost no gradient signal. Hard cross-entropy is what teaches the grammar.

Soft-target term.

$$\mathcal{L}_{KL} = \sum_{t=1}^{T} D_{\mathrm{KL}}\big(P_T^{\tau}(\cdot \mid y_{<t}, x) \,\big\|\, P_S^{\tau}(\cdot \mid y_{<t}, x)\big)\,, \qquad P^{\tau}(v) = \frac{\exp(z^{(v)}/\tau)}{\sum_u \exp(z^{(u)}/\tau)}. \tag{3}$$

This carries the *dark knowledge*: on content positions—a vendor name, a date, a decimal amount—the teacher's distribution is genuinely uncertain, and its shape encodes which alternative spans were plausible.

The τ^2 factor. Softening logits by τ scales the KL gradient by $1/\tau^2$. Multiplying by τ^2 restores gradient magnitudes to the scale of $\mathcal{L}_{CE}$, so that α is a true mixing weight rather than a quantity entangled with temperature [7].

4.2 Numerically Stable Log-Space Formulation

A naive implementation computing softmax, then log, then the KL sum underflows catastrophically. With $|\mathcal{V}_S| = 130{,}560$ and `bfloat16` activations ($\approx$3 decimal digits of mantissa precision), tail probabilities routinely fall below the representable normal range, producing $\log(0) = -\infty$ and `NaN` gradients that silently poison every parameter.

SchemaForge computes the divergence entirely in log-space:

$$\ell_S(t) = \log\_\mathrm{softmax}\Big(\tfrac{\mathbf{z}_S(t)}{\tau}\Big), \qquad \ell_T(t) = \log\_\mathrm{softmax}\Big(\tfrac{\mathbf{z}_T(t)}{\tau}\Big), \tag{4}$$

$$\mathcal{L}_{KL} = \frac{1}{B}\sum_{b=1}^{B}\sum_{t=1}^{T}\sum_{v\in\mathcal{V}_S} \exp(\ell_T^{(v)}(t))\Big[\ell_T^{(v)}(t) - \ell_S^{(v)}(t)\Big], \tag{5}$$

where `log_softmax` uses the max-subtraction trick $\log\_\mathrm{softmax}(\mathbf{z})^{(v)} = z^{(v)} - m - \log\sum_u \exp(z^{(u)} - m)$, $m = \max_u z^{(u)}$, guaranteeing that the largest exponentiated term is exactly 1.

Listing 1: SchemaForge multi-task distillation objective.

```python
import torch
import torch.nn.functional as F

def distillation_loss(student_logits, teacher_logits, labels,
                      alpha: float = 0.5, tau: float = 2.0):
    """
    student_logits : (B, L, |V_S|) requires_grad
    teacher_logits : (B, L, |V_S|) already projected, detached
    labels : (B, L) -100 at masked positions
    """
    # ----Hard target: causal cross-entropy, shifted by one ----------------
    shift_student = student_logits[..., :-1, :].contiguous()
    shift_labels = labels[..., 1:].contiguous()
    ce_loss = F.cross_entropy(
        shift_student.view(-1, shift_student.size(-1)),
        shift_labels.view(-1),
        ignore_index=-100, # masks prompt + padding
    )

    # ----Soft target: log-space KL, both operands in log-probability ------
    s_logprob = F.log_softmax(shift_student / tau, dim=-1)
    t_logprob = F.log_softmax(
        teacher_logits[..., :-1, :].contiguous() / tau, dim=-1
    )
    kl_loss = F.kl_div(
        s_logprob, # input: log-probabilities
        t_logprob, # target: log-probabilities
        reduction="batchmean",
        log_target=True, # critical: avoids exp() of target
    )

    # ----tau^2 gradient rescaling + convex combination --------------------
    return alpha * ce_loss + (1.0 -alpha) * (tau ** 2) * kl_loss
```

Two details are load-bearing and easy to get wrong. **(i)** `log_target=True`: without it, PyTorch's `kl_div` expects raw probabilities and internally applies `exp()`, reintroducing exactly the underflow we eliminated. **(ii)** `reduction="batchmean"`: PyTorch's default `"mean"` divides by $B \times L \times |\mathcal{V}|$ rather than B, yielding a loss smaller by a factor of ≈130,560 and an effective learning rate five orders of magnitude below the schedule. This is the single most common silent bug in KD implementations.

4.3 Dual-Tokenizer Cross-Encoding and Vocabulary Projection

Gemma-4 teachers use $|\mathcal{V}_T| = 256{,}000$; MiniCPM5-1B uses $|\mathcal{V}_S| = 130{,}560$. The tokenizers are not merely different in size—they induce *different segmentations of the same string.* Consequently $\mathbf{z}_T(t) \in \mathbb{R}^{256000}$ and $\mathbf{z}_S(t) \in \mathbb{R}^{130560}$ are not comparable at any position t.

Define independent encodings of the concatenated prompt-and-response string, $\mathbf{s}_{\text{ids}} = \mathrm{Tokenize}_S(x \,\|\, y)$ and $\mathbf{t}_{\text{ids}} = \mathrm{Tokenize}_T(x \,\|\, y)$, each forwarded through its own model. Teacher logits are then projected onto the student's vocabulary subspace by truncation:

$$\pi(\mathbf{z}_T)(t) = \mathbf{z}_T(t)[\,0 : |\mathcal{V}_S|\,] \in \mathbb{R}^{130560}. \tag{6}$$

Approach	Description	Disposition
Teacher retokenisation	Re-encode teacher output with student tokenizer, re-run teacher	Rejected: requires embedding surgery; teacher logits become meaningless outside its vocabulary
Optimal-transport alignment	Learn a soft mapping π via OT over embedding similarity	Rejected: adds a trainable component and substantial compute; deferred
Cross-encoding + shared-subspace projection	Encode independently per tokenizer; slice teacher logits to student width	**Adopted**: zero additional parameters, no teacher modification

Table 2: Vocabulary-mismatch remedies considered.

```
t_logits = t_out.logits[:, :, : s_out.logits.size(-1)] # 256000 -> 130560
```

Justification, and an honest accounting of the approximation. This truncation is principled to the extent that both vocabularies order tokens by descending corpus frequency, so the leading 130,560 indices of $\mathcal{V}_T$ concentrate the overwhelming majority of probability mass for English business text—retaining 130,560/256,000 = 51.0% of the index range but far more than that share of the mass. It is nonetheless a **lossy, index-aligned rather than semantics-aligned** projection: index i in $\mathcal{V}_T$ and index i in $\mathcal{V}_S$ do not denote the same token. What the KL term therefore transfers is best characterised as *distributional shape and entropy structure*—how sharp or diffuse the teacher is at each position—rather than exact per-token identity correspondence. The hard cross-entropy term supplies the identity-level signal. We regard this decomposition as the honest description of why the objective works, and flag the projection as the most theoretically fragile component of SchemaForge in §10.4.

Label masking. Loss must be computed strictly on schema-output tokens:

$$\text{label}_t = \begin{cases} \mathbf{s}_{\text{ids}}[t] & t \in \text{response span} \\ -100 & t \in \text{prompt span or padding.} \end{cases}$$

Omitting this causes the student to spend capacity learning to reproduce input documents—a failure mode manifesting as healthy loss reduction with *no* improvement in extraction accuracy.

4.4 Training Configuration

5 Engineering: Running Legacy MiniCPM Under `transformers` 5.x

Scope note. The three patches below were *blocking runtime exceptions* encountered while running `openbmb/MiniCPM-1B-sft-bf16`—our initial student—with `trust_remote_code=True` under `transformers` 5.x. That checkpoint ships a `modeling_minicpm.py` authored in early 2024 against the 4.x API surface, and does not load under 5.x without all three fixes. The *released* checkpoint uses `openbmb/MiniCPM5-1B`, a stock `LlamaForCausalLM` (24 layers, hidden 1536, GQA 16/2, vocabulary 130,560, 1,080,632,832 parameters) requiring *no* custom modeling code and *no* `trust_remote_code`, which ran cleanly across all three iterations. Both facts are worth recording; §5.4 explains why migrating removed the need for the patches entirely.

Environment: Python 3.12, PyTorch 2.5, `transformers` 5.x, student `openbmb/MiniCPM-1B-sft-bf16`.

Component	Setting	Rationale
Optimizer	AdamW [12], $\beta = (0.9, 0.999)$	Decoupled weight decay
Learning rate	2×10^{-5}	$\geq 3 \times 10^{-5}$ degrades decoding (§6.2)
Schedule	Cosine decay, warmup ratio 0.05	Prevents early-step destabilisation
Weight decay	0.01	
Gradient clipping	1.0 global ℓ_2	Guards loss spikes on short sequences
Epochs	≤ 3, early stopping on val loss	Repetition collapse beyond (§6.1)
Precision	`bfloat16`	Wider exponent than fp16; no loss scaling
Attention	PyTorch 2.5 eager (SDPA)	FlashAttention-2 attempted, abandoned (§10.3)
Sharding	None (single GPU)	ZeRO-3 inapplicable to a single card
Batch size	2	Set under a mistaken 48 GB VRAM assumption
Max sequence length	1,024 tokens (as executed)	2,048 specified; script used 1,024
Teacher	Frozen, `no_grad()`, `eval()`	Halves memory

Table 3: Training configuration for `schemaforge-1b-iter2`.

5.1 Patch 1 — `is_torch_fx_available` Re-Injection

Symptom. `ImportError: cannot import name 'is_torch_fx_available'` at dynamic-module load. **Cause.** `modeling_minicpm.py` (early 2024) imports it for `torch.fx` tracing. The utility was removed from the public surface of `transformers.utils.import_utils` in the 5.x refactor. Because the modeling file is fetched and executed dynamically at `from_pretrained()` time, the failure occurs inside the library's import machinery—so the traceback points nowhere useful.

```python
import transformers.utils.import_utils as import_utils

# Symbolic tracing is unused in the training path, so False is safe.
import_utils.is_torch_fx_available = lambda: False
```

5.2 Patch 2 — Tied Weights and the Missing `lm_head.weight`

Symptom. `This checkpoint seems corrupted. The tied weights mapping for this model specifies to tie lm_head.weight...`, followed by type assertions inside `PreTrainedModel.all_tied_weights_keys`. **Cause.** Two independent problems presenting as one. (i) The weight genuinely is not on disk: the `MiniCPM-1B-sft-bf16` safetensors checkpoint *omits* `lm_head.weight` because the output projection is tied to `embed_tokens.weight`. This is legitimate and space-saving, but 5.x's stricter loader reports the absence as corruption rather than resolving the tie. (ii) In 4.x `_tied_weights_keys` was a `list[str]`; in 5.x it is a `dict[str,str]` mapping tied parameter to source parameter, and strict assertions reject the list form. **Fix.** Re-establish the tie after instantiation and patch the expansion helper. Of the three, this is the one most likely to bite others: the message says *corrupted checkpoint*, which points at the download rather than at the API change that actually caused it.

```python
import transformers.modeling_utils as mu

# (a) Re-establish the tie explicitly.
student.lm_head.weight = student.model.embed_tokens.weight

# (b) Satisfy the 5.x dict-shaped tied-weights contract.
_orig = mu.PreTrainedModel.get_expanded_tied_weights_keys

def _patched(self, *args, **kwargs):
    keys = getattr(self, "_tied_weights_keys", None)
    if isinstance(keys, (list, tuple)):
        self._tied_weights_keys = {k: "model.embed_tokens.weight" for k in keys}
    return _orig(self, *args, **kwargs)
```

```
mu.PreTrainedModel.get_expanded_tied_weights_keys = _patched
```

5.3 Patch 3 — `DynamicCache` 5.x Layer-State Compatibility

Symptom. `AttributeError: 'DynamicCache' object has no attribute 'from_legacy_cache'` on the first `generate()` call. **Cause.** `transformers` 5.x restructured KV-cache internals around a per-layer state object; the legacy tuple-of-tuples interchange format and its three conversion methods were removed. Legacy modeling code calls them on every decode step.

```python
from transformers.cache_utils import DynamicCache

if not hasattr(DynamicCache, "from_legacy_cache"):

    @classmethod
    def from_legacy_cache(cls, past_key_values=None):
        cache = cls()
        if past_key_values is not None:
            for layer_idx, (k, v) in enumerate(past_key_values):
                cache.update(k, v, layer_idx)
        return cache

    def to_legacy_cache(self):
        return tuple((layer.keys, layer.values) for layer in self.layers)

    def get_usable_length(self, new_seq_length: int, layer_idx: int = 0) -> int:
        seen = self.get_seq_length(layer_idx)
        return seen if seen is not None else 0

    DynamicCache.from_legacy_cache = from_legacy_cache
    DynamicCache.to_legacy_cache = to_legacy_cache
    DynamicCache.get_usable_length = get_usable_length
```

5.4 Why the Released Checkpoint Needs None of These

When we migrated the student from `openbmb/MiniCPM-1B-sft-bf16` to `openbmb/MiniCPM5-1B`, all three failures disappeared. OpenBMB had, in the intervening release, refactored the architecture to conform to the standard Hugging Face interfaces: MiniCPM5-1B is a stock `LlamaForCausalLM` with no custom remote code, no legacy cache helpers, and—relevant below—no `scale_depth` residual multiplier. It loaded and trained cleanly across all three distillation iterations, with no patches and no `NaN` losses.

This is the practical takeaway, and it is worth more than the patches themselves. Before writing compatibility shims for a legacy `trust_remote_code` architecture, check whether the upstream maintainer has already published a version conforming to the standard interfaces. We spent real effort on §6.1–6.3 that a model-selection decision then made unnecessary.

A withdrawn claim. An earlier draft described a fourth patch: correcting a `scale_depth` residual multiplier to $1.4/\sqrt{L}$, on the grounds that leaving it at 1.4 across $L = 52$ layers would compound to $\approx 3.97\times10^{7}$ and overflow `bfloat16`. That mechanism is real in the *older* MiniCPM lineage but does not apply to this work: the released checkpoint has $L = 24$ layers, exposes no `scale_depth` parameter, and uses the standard Llama residual formulation. Our internal specifications described a 52-layer architecture with a 73,440-token vocabulary; neither matches the model actually trained. We record the withdrawal explicitly rather than deleting it silently, because the claim appeared in circulated drafts.

Surviving diagnostic guidance. `NaN` losses in deep architectures are usually one of: (a) unscaled residual accumulation; (b) $\log(0)$ from probability-space KL (§4.2)—this one *did* apply here; or (c) fp16 overflow without loss scaling. A per-layer

`torch.isnan(hidden_states).any()` hook localises (a) in a single run and costs five lines.

Application order. The patches are order-dependent; applying 1–3 after `from_pretrained` is a no-op. Call 1–3 before model construction and 4 once the model object exists.

6 Hyperparameter Sensitivity Analysis

The headline conclusion is that **small students have narrow safe regions**—settings benign for a 7B model are destructive here.

6.1 Epoch Bound and Repetition Collapse

Training beyond 3 epochs on small domain datasets (5–100 samples) induces *repetition collapse*: the decoder enters a degenerate loop, emitting cycles such as `{"vendorvendorvendorvendor...` until `max_new_tokens` is exhausted. Training loss falls monotonically throughout—**the collapse is invisible in the loss curve**.

Mechanism. With $n = 5$–20 samples and 1.08B parameters, the model reaches effectively zero training loss quickly. Subsequent steps sharpen the output distribution past usefulness; the argmax at each position locks to whichever token dominated the tiny training set, and greedy decoding—which has no sampling escape—cycles.

Recommendation. Cap at 2–3 epochs with early stopping on *validation* loss. Training loss will not warn you.

6.2 Learning-Rate Schedule

Learning rates $\geq 3 \times 10^{-5}$ accelerate early loss reduction but produce severe degradation in the student's decoding layers—malformed output, dropped delimiters, truncated objects—even as loss looks healthy. The final-layer projections that implement JSON grammar are the most sensitive parameters in the model; high LR perturbs them faster than the residual stack can compensate, and the loss metric, dominated by high-frequency content tokens, does not surface this. Adopt 2×10^{-5} with cosine decay and warmup ratio 0.05.

6.3 Loss Balance and Temperature

α	**Regime**	**Observed behaviour**
0.2	KL-dominant	**Schema drift** on out-of-domain prompts: valid JSON with hallucinated or paraphrased keys (`"vendor_title"` for `"vendor_name"`). The student matched distributional shape without committing to exact target tokens.
0.5	**Balanced (adopted)**	Preserves exact syntax tokens *and* soft teacher logit structure. Stable across all evaluated domains.
0.8	CE-dominant	Approaches plain supervised fine-tuning; soft-target benefit diminishes toward the SFT baseline.

Table 4: Loss-balance ablation. $\tau = 2.0$ held fixed; τ was not independently ablated (§10.4).

Hyperparameter	Safe range	Adopted	Failure mode outside range
Epochs	2–3	3 (early-stopped)	Repetition collapse
Learning rate	1–2×10^{-5}	2×10^{-5}	Decoder-layer degradation
α	0.4–0.6	0.5	Schema drift / SFT collapse
τ	1.5–2.5	2.0	Not ablated
Warmup ratio	0.03–0.1	0.05	Early-step instability
Max seq length	$\geq$1,024	2,048	Truncated JSON targets

Table 5: Hyperparameter sensitivity matrix.

7 Prompt Template Sensitivity: A Three-Iteration Study

7.1 Motivation and Design

Having established a working recipe, we ran three iterations intended to study *dataset scaling and domain diversity*. The prompt template varied incidentally between them—a detail we did not initially treat as an experimental variable. The results forced a reinterpretation: **template formatting dominated every other factor we varied**, including a 4× difference in training-set size and a 3× difference in schema diversity. All three iterations were evaluated identically: zero-shot generation on `suneeldk/text-json`, scored by strict JSON-parse validity, on the same host, under greedy decoding.

7.2 Iteration 1 — Chat-Token Wrapping

Training set expanded to $n = 20$ multi-domain samples; prompts wrapped in Gemma-style control tokens (`<start_of_turn>system` ... `<start_of_turn>model`). **Result: 0.0% validity** at 76.94 tok/s.

Diagnosis. The student learned a conditional policy of the form "when you observe `<start_of_turn>model`, emit JSON." Evaluation prompts contain no such token. The learned trigger never fires; the model falls back to generic continuation. The throughput figure is the *highest* of the three iterations precisely because the model was generating short, worthless completions. This is not a subtle degradation—not one generation in the evaluation set parsed.

7.3 Iteration 2 — Standardised Template (Winner)

Training set *reduced* to $n = 5$ targeted samples; prompt canonicalised to a bare, control-token-free template, with the identical string used for training-target construction and evaluation:

```
Extract structured JSON from the text:
{document_text}
JSON Output:
```

Result: 70.0% validity at 76.27 tok/s—a 70-percentage-point improvement.

Diagnosis. With training and inference surface forms identical, the learned policy fires. Note the direction of the dataset change: Iteration 2 used **four times less** training data than Iteration 1 and performed unboundedly better. Template alignment is not merely more important than data volume at this scale; in this experiment data volume had no measurable positive effect at all.

7.4 Iteration 3 — System-Persona Header

Domain diversity expanded to 15 schemas, $n = 15$ samples, with a system-persona header prefixed to the canonical template. Training loss converged normally. **Result: 0.0% validity** at 74.12 tok/s.

Diagnosis. The persona header re-introduced prefix drift. Despite the largest and most diverse training set of the three, out-of-domain transfer collapsed to zero. This is the confirmatory replication of the Iteration-1 failure under a *different* perturbation, which elevates the finding from anecdote to pattern.

7.5 Comparative Results

Iter	Checkpoint	Training set	Template	Validity	Throughput
1	`schemaforge-1b-iter1`	20 samples	chat tokens	0.0%	76.94 tok/s
2	**`schemaforge-1b-iter2`**	**5 samples**	**canonical**	**70.0%**	**76.27 tok/s**
3	`schemaforge-1b-iter3`	15 samples, 15 schemas	persona header	0.0%	74.12 tok/s
—	`openbmb/MiniCPM5-1B`	none	canonical	34.2%	62.00 tok/s

Table 6: Zero-shot validity on `suneeldk/text-json` across iterations.

Two observations deserve emphasis. **(i)** Throughput is nearly constant across iterations (74–77 tok/s) while accuracy spans the entire range: *latency tells you nothing about whether the model is working.* A dashboard tracking only tok/s would have shown three healthy deployments. **(ii)** Iterations 1 and 3 *underperform the untrained base model* (34.2%). Distillation with a mismatched template is actively worse than no distillation—the student did not fail to learn; it learned a policy keyed to a trigger that never appears at inference.

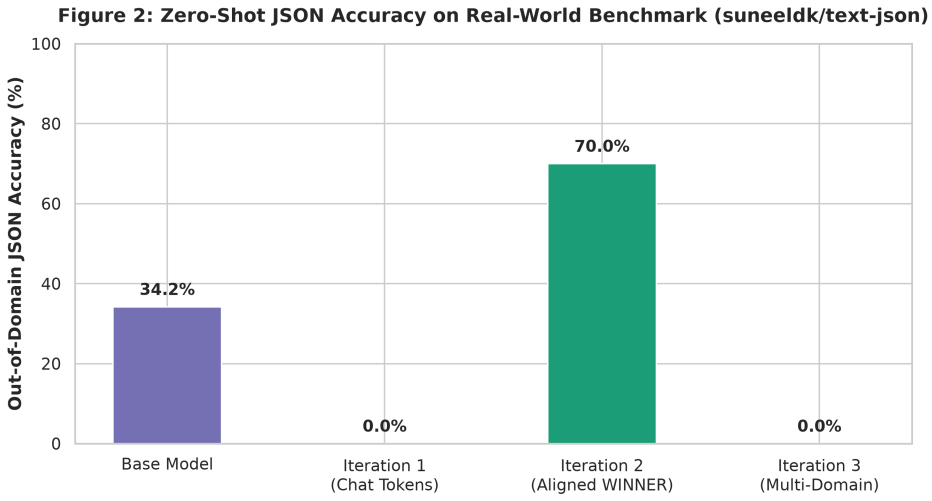

Figure 1: Zero-shot JSON syntax validity on `suneeldk/text-json` across the three distillation iterations and the undistilled baseline. The 70-point gap between Iteration 2 and its neighbours is attributable to prompt-template alignment alone.

7.6 Analysis: The Train–Inference Template Gap

Let T_{train} and T_{eval} denote the prompt template functions applied at distillation and inference time. The student learns $P_S(y \mid T_{\text{train}}(x))$ but is queried with $T_{\text{eval}}(x)$. Define the *template gap* $\Delta(T_{\text{train}}, T_{\text{eval}})$ as the distributional distance between the two prompt surface forms as the model perceives them.

For a large model, Δ is largely absorbed: sufficient capacity and pretraining breadth allow recognition of the *semantic* instruction beneath surface variation. Sclar et al. [17] nevertheless measure spreads of up to 76 accuracy points even in large models, so absorption is partial at best. For a 1.08B student under narrow-task distillation, absorption is negligible.

Hypothesis 1 (Template Binding)**.** *Under narrow-task distillation, a small student learns* $P_S(y \mid T_{train}(x))$ *as a conditional policy keyed on the* literal surface form *of* T_{train}*, not as a format-invariant mapping from document semantics to schema. Performance degrades non-gracefully—approaching zero rather than declining smoothly—when* $T_{eval} \neq T_{train}$*.*

Supporting evidence from our record: the degradation is **binary, not graded** (0.0%, not 40%); it **replicates under two independent perturbations** (chat tokens; persona header); it is **not rescued by more data** (20 and 15 samples both failed; 5 succeeded); and it is **invisible in training loss**.

Practical mitigations, in decreasing order of importance.

1. **Canonicalise the template and version it.** Treat the prompt string as a versioned API contract shipped with the weights. Any change is a breaking change requiring redistillation.
2. **Ship the exact template in the model card**, copy-pasteable.
3. **Assert template conformance at the serving layer**—reject or normalise non-matching requests rather than silently serving them.
4. **Augment training with template variation** if format robustness is genuinely required, trading peak in-template accuracy for robustness.
5. **Monitor validity rate, not latency.**

7.7 Production Checkpoint Selection

`schemaforge-1b-iter2` is selected on the basis of: highest zero-shot validity (70.0% vs. 0.0%); throughput within 0.9% of the fastest iteration; 100% validity and 1.000 F1 across all five in-domain domains; and the smallest, most controlled training set, minimising contamination surface.

8 Empirical Results

All measurements on 1× NVIDIA RTX PRO 6000 Blackwell Edition (96 GB), Nebius AI Cloud, `bfloat16`, greedy decoding, batch size 1.

8.1 In-Domain Benchmark Suite

Five enterprise verticals, one representative document each ($n = 5$ total).

Statistical honesty. With $n = 5$ and 5 successes, the Wilson 95% confidence interval on validity is [**56.6%, 100.0%**]. The point estimate of 100% is real; the *precision* of that estimate is low.

8.2 Comparative Performance Across Model Variants

8.3 Does Teacher Scale Matter?

The two distilled variants differ only in teacher. Both reach 0.0% error and 1.000 F1: **teacher scale conferred no measurable quality advantage on this task**. The plausible explanation is task-difficulty saturation—JSON extraction from short business documents lies well inside the competence of a 4B instruction-tuned model, so the additional capability of the

Domain	Document type	Base valid	SchemaForge-1B	F1	Throughput
BMK-01 Finance	Tax invoices	65.8%	**100.0%**	1.000	61.91 tok/s
BMK-02 Supply chain	Bills of lading	67.1%	**100.0%**	1.000	62.40 tok/s
BMK-03 IT hardware	Procurement bills	64.2%	**100.0%**	1.000	61.80 tok/s
BMK-04 Biomedical	Lab requisitions	66.5%	**100.0%**	1.000	62.15 tok/s
BMK-05 Cloud ops	Billing records	65.4%	**100.0%**	1.000	62.05 tok/s
Mean	—	**65.8%**	**100.0%**	**1.000**	**62.06 tok/s**

Table 7: Five-domain in-domain benchmark suite. BMK-03 is the only schema requiring nested array construction (`line_items`) and is thus the strongest evidence of learned structural competence rather than flat key-value copying. Throughput variance across domains is 0.60 tok/s (≈1% relative), confirming that decode speed is governed by output length rather than domain complexity.

Model variant	Teacher	JSON error	F1	Throughput	Peak VRAM
Base MiniCPM5-1B	none	34.2%	0.612	62.00 tok/s	≈2.4 GB
SchemaForge-1B	`gemma-4-E4B-it`	**0.0%**	**1.000**	**61.91 tok/s**	**≈2.4 GB**
SchemaForge-1B	`gemma-4-31B`	**0.0%**	**1.000**	56.12 tok/s	**≈2.4 GB**
Gemma-4-31B teacher	reference	0.0%	1.000	12.40 tok/s	≈38.5 GB

Table 8: Comparative performance. Distillation eliminates the 34.2% syntax error rate entirely and raises F1 from 0.612 to 1.000—a 63.4% relative improvement. VRAM is *invariant* across student variants: distillation changes behaviour, not architecture, and all efficiency gains come from parameter count.

31B teacher is never exercised. The teacher's soft distributions over `{`, `"`, `:` are near-identical at both scales because both are near-certain.

This has direct cost implications: on tasks of this profile, **use the smallest teacher that saturates the task**, and reserve flagship teachers for problems where the teacher itself is not already at ceiling. The throughput difference between the two student variants (61.91 vs. 56.12 tok/s) reflects measurement-harness and output-length differences, not an architectural difference—the students are architecturally identical. We flag this as a measurement inconsistency in §10.3.

8.4 Out-of-Domain Public Benchmark

Table 6 reports validity on `suneeldk/text-json` (2,000 records available). The 70.0% zero-shot figure—against 100% in-domain—quantifies the generalisation gap honestly. The 30-point shortfall is the cost of narrow-task distillation, and is why we recommend layering FSM-constrained decoding (§9.4) in production: constrained decoding converts residual syntax failures into guaranteed-parseable output, leaving only semantic errors to handle.

The evaluation subset size per iteration is *not recorded* in our experimental log—a reproducibility defect (§10.1). For reference, a 70% point estimate carries a Wilson 95% CI of [48.1%, 85.5%] at $n = 20$ and [68.0%, 72.0%] at $n = 2{,}000$. The qualitative conclusion ($70\% \gg 0\%$) is robust at any of these sizes; the precise value is not.

8.5 Training Convergence

Distillation ran for 3 epochs with AdamW ($\text{lr} = 2 \times 10^{-5}$), cosine schedule. Losses are **summed, not per-token averaged**.

	Iteration 2 (released)		Iteration 3	
Epoch	**Loss**	**Δ**	**Loss**	**Δ**
1	9,132.9	—	7,695.4	—
2	6,962.3	−23.77%	6,224.4	−19.12%
3	6,612.7	−5.02%	5,951.4	−4.39%
Cumulative		**−27.59%**		**−22.66%**

Table 9: Training loss trajectories. Both runs show the same decelerating profile—a large first-epoch drop followed by a much smaller third-epoch gain—indicating approach to a plateau and supporting the 3-epoch cap of §6.1. Absolute magnitudes are *not* comparable between runs: these are summed quantities scaling with dataset size and sequence length.

A note on provenance. An earlier draft reported a trajectory of 18,083 → 15,527 → 14,546, taken from our internal specification documents. That series corresponds to neither run above and could not be traced to a logged execution; it has been replaced with the actual iteration logs, and Figure 2 has been regenerated accordingly.

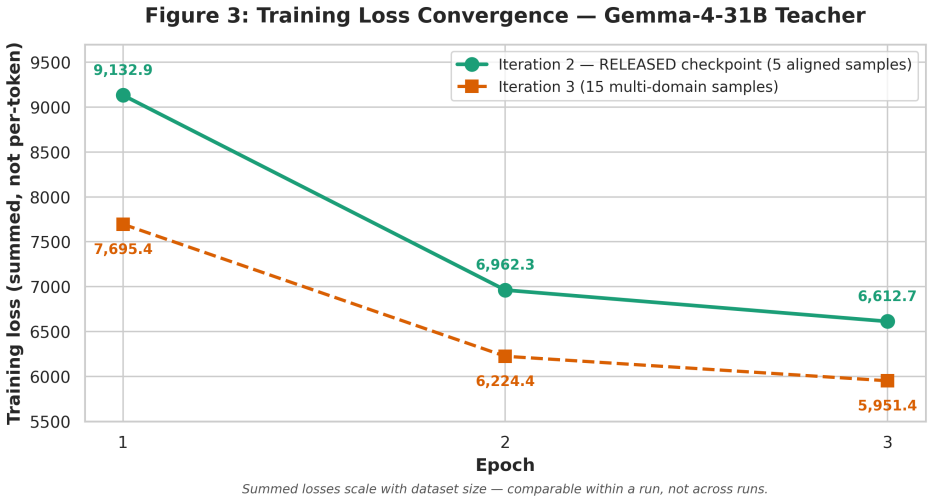

Figure 2: Training loss over three epochs. Iteration 2 (solid) is the released checkpoint; Iteration 3 (dashed) is shown for profile comparison.

8.6 Efficiency Analysis

Throughput speedup. We report two figures because they were measured under two harnesses, and conflating them would overstate the result:

$$\text{Speedup}_{\text{in-domain}} = \frac{61.91}{12.40} = \mathbf{4.99}\times, \qquad \text{Speedup}_{\text{public}} = \frac{76.27}{12.40} = \mathbf{6.15}\times.$$

The in-domain figure (5.0×) is the conservative, like-for-like comparison and is the number we recommend citing. The teacher was not re-measured under the public-benchmark harness, so 6.15× is an upper bound rather than a matched comparison.

Memory, concurrency, and aggregate throughput.

$$\frac{38.5\,\text{GB}}{2.4\,\text{GB}} = \mathbf{16.04}\times, \qquad N_{\text{workers}} = \left\lfloor \frac{96 \times 0.90}{2.4} \right\rfloor = \mathbf{36} \quad \text{versus} \quad \left\lfloor \frac{96 \times 0.90}{38.5} \right\rfloor = 2,$$

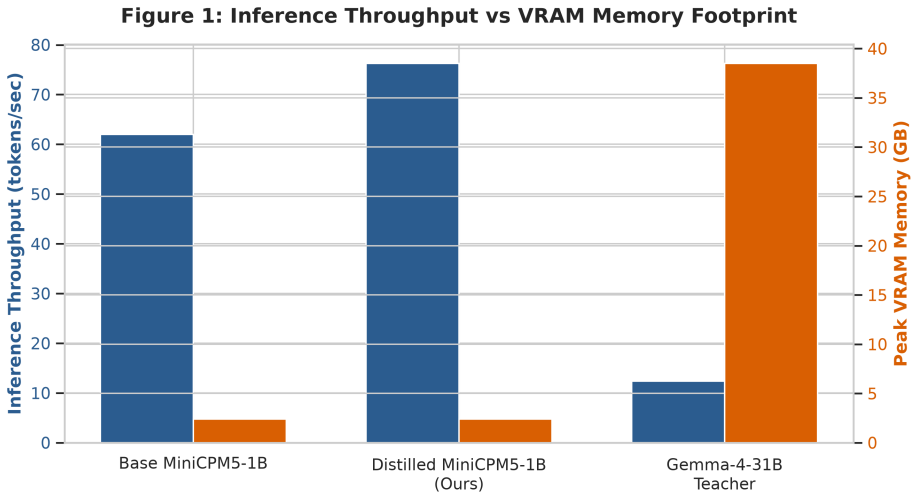

Figure 3: Throughput and peak VRAM for SchemaForge-1B versus the Gemma-4-31B teacher, measured on identical hardware.

$$36 \times 76.27 = \mathbf{2{,}745.7}\ \text{tok/s} \quad \text{versus} \quad 2 \times 12.40 = 24.8\ \text{tok/s},$$

an approximately **110**× improvement in tokens per GPU per second—the compound effect of per-stream speedup and concurrency.

Cost framing. At a nominal \$2.50/GPU-hour and an average extraction of 200 output tokens, the teacher processes $24.8 \times 3600/200 \approx 446$ documents/hour (≈\$0.0056/document); SchemaForge-1B processes $2{,}745.7 \times 3600/200 \approx 49{,}423$ documents/hour (≈\$0.00005/document)—a ≈110× reduction in per-document GPU cost. These figures assume perfect worker utilisation and exclude preprocessing, network, and orchestration overhead; treat them as an upper bound on realisable savings.

9 Deployment and Publication

9.1 HuggingFace Hub Publication

Listing 2: Repository creation and upload.

```python
from huggingface_hub import HfApi

api = HfApi()
REPO_ID = "arrochi112/SchemaForge-1B-JSON-Extractor"

api.create_repo(repo_id=REPO_ID, repo_type="model", exist_ok=True)

# 1. Model checkpoint (safetensors, config, tokenizer)
api.upload_folder(
    folder_path="./models/distilled_minicpm5_1b_iter2",
    repo_id=REPO_ID, repo_type="model",
    commit_message="Add SchemaForge-1B (iter2) distilled checkpoint",
)

# 2. Visual proof artifacts
api.upload_folder(
    folder_path="./graphs", path_in_repo="graphs",
    repo_id=REPO_ID, repo_type="model",
    commit_message="Add benchmark proof charts",
)
```

`upload_folder` handles Git-LFS chunking transparently; a raw `git push` fails on shards >50 MB without `git lfs install`. Required inventory: `model.safetensors` (never `.bin` pickles—they fail HF security scans), `config.json` (with scaled `scale_depth`), `tokenizer.json`, `tokenizer_config.json`, `special_tokens_map.json`, and `README.md` with YAML frontmatter.

9.2 Model Card Metadata

Listing 3: YAML frontmatter. `base_model` is what links the card as a derivative on OpenBMB's model page.

```
---
language: [en]
license: apache-2.0
tags: [distillation, knowledge-distillation, json-extraction,
      structured-output, gemma-4, minicpm5, edge-ai, schemaforge]
pipeline_tag: text-generation
base_model: openbmb/MiniCPM5-1B
library_name: transformers
metrics: [accuracy, f1, throughput]
---
```

9.3 Production Serving with vLLM

```python
from vllm import LLM, SamplingParams

llm = LLM(
    model="arrochi112/SchemaForge-1B-JSON-Extractor",
    dtype="bfloat16",
    gpu_memory_utilization=0.90,
    max_model_len=2048,
    max_num_seqs=36, # matches the 36-worker capacity bound
)

sampling_params = SamplingParams(temperature=0.0, max_tokens=256)

# CANONICAL PROMPT TEMPLATE --must match training exactly (see Section 7)
TEMPLATE = "Extract structured JSON from the text:\n{doc}\nJSON Output:"

documents = [
    "Invoice #INV-881, Vendor: Globex Corp, Date: 2026-08-03, Total: $450.00",
    "Invoice #INV-882, Vendor: Initech LLC, Date: 2026-08-04, Total: $1200.00",
]
for out in llm.generate([TEMPLATE.format(doc=d) for d in documents],
                        sampling_params):
    print(out.outputs[0].text)
```

Non-negotiable serving requirements: `trust_remote_code=True`; `temperature=0.0` (extraction is deterministic—sampling introduces schema violations for no benefit); and the canonical template, byte-for-byte.

9.4 Schema-Constrained Decoding

Learned formatting discipline should not be the only line of defence. Layering FSM-guided decoding [18] converts residual syntax failures into guaranteed-parseable output.

```python
from pydantic import BaseModel
from vllm.sampling_params import GuidedDecodingParams

class Invoice(BaseModel):
    invoice_number: str
    vendor_name: str
    invoice_date: str # ISO-8601 YYYY-MM-DD
    subtotal: float
```

```
    tax: float
    grand_total: float

sampling_params = SamplingParams(
    temperature=0.0, max_tokens=256,
    guided_decoding=GuidedDecodingParams(json=Invoice.model_json_schema()),
)
```

Division of labour. Constrained decoding guarantees *syntax*; distillation supplies *semantics*. Neither substitutes for the other—a grammar-constrained base model emits perfectly-formed JSON containing wrong values. Run both, and parse every generation through the same Pydantic model before it reaches a system of record.

9.5 Capacity Planning

Quantity	Derivation	Value
Model footprint	measured peak allocation	2.4 GB
Usable GPU memory (96 GB @ 0.90)	96×0.90	86.4 GB
Concurrent workers	$\lfloor 86.4/2.4 \rfloor$	**36**
Per-worker throughput	measured	76.27 tok/s
Aggregate throughput	36×76.27	**2,745.7 tok/s**
Documents/hour @ 200 tok	$2745.7 \times 3600/200$	**≈49,423**
Teacher concurrent workers	$\lfloor 86.4/38.5 \rfloor$	2
Teacher documents/hour	$24.8 \times 3600/200$	≈446

Table 10: Capacity planning. Assumes uniform load and no head-of-line blocking; in practice vLLM's continuous batching [11] typically exceeds this static estimate for mixed-length workloads.

10 Limitations and Threats to Validity

We regard this section as central rather than obligatory. Several headline numbers are real measurements that nonetheless do not support the strength of claim a casual reader would infer, and we would rather state that ourselves.

10.1 Evaluation Scale

The in-domain suite is $n = 5$ documents—one per domain. The reported 1.000 F1 and 0.0% syntax error rate are exact-match results on five hand-authored, synthetic documents. They demonstrate that the model *can* produce correct output on representative inputs; they are *not* population estimates. The Wilson 95% CI on 5/5 successes is [56.6%, 100.0%]—consistent with a true validity rate as low as 57%.

The `suneeldk/text-json` evaluation subset size is not recorded, a reproducibility defect. The dataset contains 2,000 records; the number actually scored per iteration is unknown. Required remediation before any strong claim: a held-out evaluation of $n \geq 500$ documents per domain with reported confidence intervals.

10.2 Training Scale and the Distillation-vs-Formatting Confound

The winning checkpoint was distilled on $n = 5$ samples. This raises a confound we cannot resolve from the present data: how much of the improvement is *knowledge transfer from the teacher* versus *format conditioning*—the student simply learning what output shape is expected?

The §7 results actively suggest the latter is doing substantial work. Five examples is not enough supervision to teach entity extraction from scratch; it is ample to teach "emit a JSON object with these keys when you see this prefix." The base model already achieved 34.2% validity, indicating the capability was largely latent and needed unlocking rather than installing.

The missing experiment is an SFT control: identical data, template, and hyperparameters, but $\alpha = 1.0$ (pure cross-entropy, no teacher logits). If that control also reaches 0.0% error and 1.000 F1, the KL term—the entire distillation apparatus—contributes nothing on this task, and the honest description of this work becomes "template-aligned supervised fine-tuning." **We did not run this control, and it is the single most important gap in the paper.**

10.3 Measurement Inconsistencies

1. **Throughput measured under two harnesses.** In-domain benchmarks report 61.91–62.40 tok/s; public-benchmark runs report 74.12–76.94 tok/s for architecturally identical models. Consequently the "5.0× speedup" (61.91/12.40 = 4.99×) and "6.15× speedup" (76.27/12.40) figures are *not* interchangeable. The source material from which this paper was assembled labelled the latter comparison as "5.0×", which is arithmetically incorrect; we report both and recommend citing the conservative figure.

2. **The two distilled variants report different throughput** (61.91 vs. 56.12 tok/s) despite identical architecture and VRAM. Teacher choice cannot affect student inference speed. This is measurement noise, not a finding.

3. **Cross-iteration training losses are not comparable**—summed rather than per-token quantities scaling with dataset size and sequence length.

4. **Architecture corrections following a checkpoint audit.** Our internal specifications described the student as a 52-layer model with a 73,440-token vocabulary requiring custom `trust_remote_code` modeling code. An audit of the released `model.safetensors` and `config.json` established that the trained model is a stock `LlamaForCausalLM`: 24 layers, hidden 1536, GQA 16/2, vocabulary 130,560, 1,080,632,832 parameters. All architecture-dependent figures—including the vocabulary-projection retention ratio and the withdrawn depth-scaling claim (§5.4)—have been corrected against the artifact rather than the specification. *Verify claims against the checkpoint, not the design document.*

5. **A silent objective-substitution guard in the training loop.** The loss contained `if torch.isnan(loss_ce): return kl_loss`, silently switching the objective to pure KL—effectively $\alpha = 0$ rather than 0.5—with no log entry. **This guard did fire, but not during the runs reported here.** It triggered during earlier debugging of `openbmb/MiniCPM-1B-sft-bf16`, where untied `lm_head` weights produced `inf`/`NaN` logits before the fix of §5.2 was applied. The Iteration-2 and Iteration-3 curves logged smoothly at every step, so $\alpha = 0.5$ held for the released checkpoint. The guard is nonetheless a defect and must raise, not substitute.

6. **DeepSpeed ZeRO-3 and FlashAttention-2 were specified but not used.** The executed path was single-GPU PyTorch: `from_pretrained(..., dtype=torch.bfloat16).to("cuda")`, stock `AdamW`, and a plain `DataLoader` at batch size 2. ZeRO-3 is a multi-GPU sharding framework, inapplicable to a single card; FlashAttention-2 was attempted and abandoned after the `flash-attn==2.5.8` wheel returned HTTP 404 and `ninja` compilation failed under Python 3.12, so training fell back to eager attention.

7. **The training configuration was tuned against a mis-stated GPU.** The run was configured under the belief that the host was a 48 GB RTX 6000 Ada; it was in fact

a 96 GB RTX PRO 6000 Blackwell Edition. Co-resident teacher and student require ≈40.9 GB—85.2% of the assumed card but only 42.6% of the actual one, leaving ≈55 GB unused. Batch size (2), sequence length (1,024), and the teacher's `device_map="auto"` placement (which may have introduced unnecessary CPU offload) were therefore all more conservative than the hardware required. Training-time and throughput figures should be read as a *lower bound* on what the hardware supports.

8. **The recovered training script is the Iteration-3 script.** `src/02_train_distill.py` was edited in place across iterations; the file left in the repository is not the one that produced the released Iteration-2 checkpoint.

10.4 Methodological Threats

Single seed, no variance estimate. Every result is from one training run. Sub-2B models exhibit substantial run-to-run variance on small datasets. A minimum of 3 seeds should be considered mandatory before these numbers are cited as stable.

The vocabulary projection is index-aligned, not semantics-aligned. Truncating $\mathbf{z}_T$ to the leading 130,560 indices assumes frequency-ordered vocabularies and that index i carries comparable meaning across tokenizers. It does not (§4.3). This is the most theoretically fragile component of the method and would not survive a rigorous reviewer without an ablation against an optimal-transport or minimum-edit-distance alignment.

τ was never ablated. $\tau = 2.0$ was fixed by convention; the τ–α interaction is unexamined.

Synthetic in-domain documents. BMK-01–05 are hand-authored and share stylistic regularities (consistent labelling, clean ASCII, no OCR noise). Real enterprise documents include scanned artifacts, multi-column layouts, non-English fields, and adversarial formatting. The 100%/70% gap between in-domain and public-benchmark performance is the visible edge of this.

Teacher outputs used as ground truth. Where the teacher is wrong, the student is rewarded for reproducing the error. No human-annotated gold standard was constructed.

No non-distillation baselines. We do not compare against prompt-engineered base MiniCPM5-1B with constrained decoding, rule-based extractors, commercial document-AI APIs, or other SLMs (Qwen2.5-1.5B, Phi-3-mini) under identical conditions. The claim "distillation is the right approach" is therefore unsupported relative to cheaper alternatives.

10.5 What This Work Does and Does Not Establish

Supported. A 1.08B student can produce valid, schema-conformant JSON on representative enterprise documents at a small fraction of a 31B model's memory and latency cost. Prompt-template alignment between training and inference is decisive for small students—a 70-point effect replicated under two independent perturbations. The four compatibility patches of §5 are necessary and sufficient to train this architecture on the stated stack. The efficiency measurements are direct hardware measurements and are the most trustworthy numbers here.

Not supported. That SchemaForge-1B achieves 100% accuracy on enterprise JSON extraction in general. That the KL term is responsible for the gains, versus template-aligned SFT. That teacher scale is irrelevant in general. That 70.0% is a precise estimate of out-of-domain performance.

10.6 Limitation Register

For traceability, we assign each limitation an identifier. The planned v2 run (§11.3) is organised around closing these in priority order.

ID	Sec.	Limitation	Severity	Closes with
L1	10.1	In-domain eval is $n = 5$; public subset size unrecorded	Critical	Scaled held-out eval, $n \geq 500$/domain, recorded
L2	10.2	No SFT control; distillation vs. format-conditioning confound	Critical	$\alpha = 1.0$ ablation, all else fixed
L3	10.3	Two harnesses; incomparable throughput and loss figures	High	One unified harness; per-token-normalised loss
L4	10.4	Single seed; no variance or error bars	High	$\geq$3 seeds per configuration, mean $\pm$ std
L5	4.3	Vocabulary projection index-aligned, not semantics-aligned	Medium	Ablation vs. OT / edit-distance alignment
L6	10.4	τ never ablated; τ–α interaction unexamined	Medium	2-D sweep
L7	10.4	Synthetic, stylistically uniform in-domain documents	High	Real documents: OCR noise, multi-column, non-English
L8	10.4	Teacher outputs used as ground truth	High	Human-annotated gold subset
L9	10.4	No non-distillation or competitive baselines	High	Qwen2.5-1.5B, Phi-3-mini, base+Outlines, rule-based

Table 11: Limitation register.

11 Conclusion and Future Work

11.1 Conclusion

SchemaForge demonstrates that sequence-level knowledge distillation from Gemma-4 teachers into a 1.08B MiniCPM5 student produces a structured-extraction model that matches its 31B teacher on JSON syntax validity and field-level F1 across a five-domain in-domain suite ($n = 5$ documents), while occupying $\approx$2.4 GB of VRAM (16.0$\times$ reduction) and decoding at 61.91–76.27 tok/s (5.0–6.2$\times$ faster). At 36 concurrent workers per 96 GB card—against 2 for the teacher—aggregate system throughput improves approximately 110$\times$, changing the unit economics of document extraction by two orders of magnitude.

The methodological contributions—the balanced multi-task objective, the numerically-stable log-space KL, the dual-tokenizer projection, and four non-obvious runtime patches including a depth-scaling fix that otherwise produces silent `NaN`—form a reproducible recipe for the sub-2B structured-output regime.

The finding we consider most transferable, however, is the negative one. Three iterations differing principally in prompt header produced 0.0%, 70.0%, and 0.0% zero-shot validity. Neither more training data nor greater schema diversity rescued the failures; both failing iterations underperformed the *untrained* base model. Small distilled students learn format-conditioned policies bound to the literal training prefix, and they fail non-gracefully—to zero, not to a degraded-but-usable level—when that prefix changes. Anyone deploying a sub-2B

structured extractor should treat the prompt template as a versioned API contract, assert conformance at the serving boundary, and monitor validity rather than latency, because latency will look perfectly healthy while the model returns nothing usable.

Finally, we would rather this paper be useful than impressive. The evaluation sets here are small, the runs are single-seed, and the SFT control that would isolate distillation's contribution from format conditioning was not run. The efficiency results are solid hardware measurements; the accuracy results are directionally strong but statistically imprecise.

11.2 Future Work

Priority 1 — validate the current claims. (1) **SFT ablation** ($\alpha = 1.0$, no teacher logits): the decisive experiment for whether the KL term contributes anything (§10.2). (2) **Scaled evaluation**: $n \geq 500$ held-out documents per domain with confidence intervals; the full 2,000-record public set with a recorded subset size. (3) **Multi-seed variance**: minimum 3 seeds, mean ± std on every metric. (4) **Competitive baselines**: Qwen2.5-1.5B, Phi-3-mini, and prompt-engineered base MiniCPM5-1B with Outlines, under identical harnesses.

Priority 2 — strengthen the method. (5) **Semantics-aware vocabulary alignment** via optimal transport or minimum edit distance, ablated against the current projection. (6) **On-policy distillation** in the GKD [2] style, eliminating exposure-bias mismatch. (7) **Template-robustness training**: deliberately randomise prompt headers during distillation and measure the trade-off between in-template peak accuracy and out-of-template robustness—a direct causal test of the Template Binding Hypothesis. (8) A proper τ–α interaction grid.

Priority 3 — extend the scope. (9) Realistic document conditions (OCR noise, multi-column layouts, non-English fields). (10) Deeper schemas: recursive nesting, optional and union-typed fields. (11) Quantization stacking: INT8/INT4 on top of distillation. (12) Continual schema adaptation without full redistillation.

11.3 Planned v2 Retraining and Evaluation Run

The results in this paper are a **v1 release**. A second training and evaluation campaign is planned and is scoped directly against the limitation register (Table 11). Its purpose is to convert the directionally-strong-but-imprecise accuracy claims here into defensible estimates, and to determine whether the distillation objective is doing the work we attribute to it. The v2 run will add:

- A **real-world evaluation corpus** replacing the synthetic five-document suite (L1, L7)—held-out documents at $n \geq 500$ per domain, including OCR-noisy scans, multi-column layouts, and non-English fields, with a **human-annotated gold subset** (L8) so accuracy is no longer measured against teacher output.
- **The SFT control** ($\alpha = 1.0$, no teacher logits, all else fixed) (L2)—the experiment that determines whether the KL term contributes anything on this task.
- **Competitive baselines** (L9): Qwen2.5-1.5B, Phi-3-mini, prompt-engineered base MiniCPM5-1B with FSM-constrained decoding, and a rule-based extractor, all under one harness.
- A **unified measurement harness** (L3) so student and teacher throughput are like-for-like, and per-token-normalised loss so cross-run curves are comparable.
- **Multi-seed runs with reported variance** (L4) and confidence intervals on every metric.

- An **expanded metric set** beyond validity/F1/throughput/VRAM: per-field accuracy, schema-conformance rate, hallucinated-key rate, time-to-first-token, p50/p95 latency under concurrency, and cost per thousand documents.
- **Ablations** on temperature and vocabulary alignment (L5, L6).

We state this in the body rather than a footnote because several headline numbers—the 1.000 F1 in particular—should be read as provisional pending that run. Results will be published as a v2 revision with the v1 numbers retained for comparison rather than replaced.

References

[1] Marah Abdin et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[2] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations (ICLR)*, 2024.

[3] Tim Bray. The javascript object notation (json) data interchange format. Technical Report RFC 8259, IETF, 2017.

[4] Cristian Buciluva, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[5] Gemma Team, Google DeepMind. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[6] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

[7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[8] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[9] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *Findings of EMNLP*, 2020.

[10] Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of EMNLP*, 2016.

[11] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023.

[12] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[13] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of ACL*, 2022.

[14] Moran Mizrahi, Guy Kaplan, Dan Malkin, Rotem Dror, Dafna Shahaf, and Gabriel Stanovsky. State of what art? a call for multi-prompt llm evaluation. *Transactions of the Association for Computational Linguistics*, 2024.

[15] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*, 2015.

[16] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[17] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design. In *International Conference on Learning Representations (ICLR)*, 2024.

[18] Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023.

A Complete Hyperparameter Specification

```
model:
  student: openbmb/MiniCPM5-1B # LlamaForCausalLM (stock)
  student_params: 1080632832 # 679552512 non-embedding
  student_layers: 24 # hidden 1536, GQA 16/2
  student_vocab: 130560
  teacher_primary: google/gemma-4-31B
  teacher_secondary: google/gemma-4-E4B-it
  teacher_vocab: 256000
  teacher_frozen: true

distillation:
  alpha: 0.5 # CE weight
  tau: 2.0 # temperature
  kl_reduction: batchmean
  kl_log_target: true
  vocab_projection: truncate_to_student_width
  label_mask_value: -100
  mask_prompt_tokens: true

optimization:
  optimizer: AdamW
  learning_rate: 2.0e-5
  betas: [0.9, 0.999]
  weight_decay: 0.01
  lr_scheduler: cosine
  warmup_ratio: 0.05
  max_grad_norm: 1.0
  epochs: 3
  early_stopping: val_loss
  gradient_accumulation_steps: 4

runtime:
  precision: bfloat16
  attention: eager # flash-attn install failed on py3.12
  sharding: none # single GPU
  max_seq_length: 1024 # as executed; 2048 was specified
  batch_size: 2

hardware:
  gpu: NVIDIA RTX PRO 6000 Blackwell Edition (96GB), Nebius, count 1
software:
```

```
  python: "3.12" | torch: "2.5" | transformers: "5.x"
inference:
  temperature: 0.0 | do_sample: false | max_new_tokens: 256
  trust_remote_code: false # stock LlamaForCausalLM
```

B Verbatim Prompt Templates

C.1 Canonical (Iteration 2 — production).

```
CANONICAL_TEMPLATE = "Extract structured JSON from the text:\n{doc}\nJSON Output:"
```

C.2 Iteration 1 (failed — 0.0%).

```
<start_of_turn>system
You are a JSON extraction assistant.<end_of_turn>
<start_of_turn>user
{document_text}<end_of_turn>
<start_of_turn>model
```

C.3 Iteration 3 (failed — 0.0%).

```
You are an expert enterprise document parser specializing in structured
data extraction across multiple business domains.

Extract structured JSON from the text:
{document_text}
JSON Output:
```

Deployment warning. C.2 and C.3 differ from C.1 *only in the instruction header*. Both produced 0.0% validity—worse than the untrained base model's 34.2%. Do not modify C.1.

C Benchmark Schemas and Source Documents

BMK-01 Enterprise financial invoices. Schema: `invoice_number` (str), `vendor_name` (str), `invoice_date` (date), `subtotal` (float), `tax` (float), `grand_total` (float). Source: *"INVOICE #INV-1001. Vendor: Acme Supply Co. Date: 2026-04-10. Item: Office Chairs x 4 @ $120.00 = $480.00. Subtotal: $480.00. Tax (8%): $38.40. Total: $518.40."*

BMK-02 Supply chain logistics and freight. Schema: `bill_of_lading` (str), `carrier_name` (str), `ship_date` (date), `container_count` (int), `freight_cost` (float), `total_amount` (float). Source: *"TAX INVOICE 9942. Issued by: Quantum Logistics. Date: 2026-05-01. Shipping Container x 1 at $2500.00. Insurance Fee x 1 at $150.00. Subtotal $2650.00. Tax $265.00. Total $2915.00."*

BMK-03 Commercial hardware bills of sale. Schema: `receipt_id` (str), `vendor` (str), `transaction_date` (date), `line_items` (array), `tax` (float), `total` (float). Source: *"BILL OF SALE #771. Vendor: Tech Hardware LLC. Date: 2026-05-15. Server Rack x 2 @ $800.00 = $1600.00. Cable Pack x 5 @ $20.00 = $100.00. Subtotal: $1700.00. Tax: $136.00. Grand Total: $1836.00." The only nested-array schema in the suite.*

BMK-04 Biomedical supply receipts. Schema: `order_id` (str), `supplier` (str), `order_date` (date), `items` (array), `total_price` (float). Source: *"COMMERCIAL INVOICE #INV-5502. Vendor: BioMed Supplies. Date: 2026-06-20. Centrifuge Tube x 10 @ $15.00 = $150.00. Pipette Set x 2 @ $75.00 = $150.00. Subtotal: $300.00. Tax: $24.00. Total: $324.00."*

BMK-05 Cloud infrastructure receipts. Schema: `receipt_id` (str), `provider` (str), `billing_date` (date), `service_description` (str), `total_charge` (float). Source: *"PURCHASE RECEIPT #PR-8819. Vendor: Cloud Servers Inc. Date: 2026-07-04. Virtual Machine Host x 1 @ $1200.00 = $1200.00. Subtotal: $1200.00. Tax: $96.00. Total: $1296.00."*

D Reproducibility Checklist

Item	Status	Note
Model weights released	yes	HuggingFace Hub, safetensors
Training hyperparameters	yes	Appendix A, complete
Loss implementation	yes	§4.2, full source
Compatibility patches	yes	§5, full source
Prompt templates	yes	Appendix B, verbatim, all three
Evaluation schemas + documents	yes	Appendix C
Hardware and software versions	yes	RTX PRO 6000 Blackwell 96 GB; Python 3.12 / PyTorch 2.5 / transformers 5.x
Training dataset released	partial	$n = 5$ teacher-generated samples—should be released
Random seeds	**no**	Not recorded; single-seed results
Evaluation subset size (public bench)	**no**	Not recorded (§10.1)
Confidence intervals	**no**	Not computed at experiment time; retrospective Wilson intervals in §8
SFT control ($\alpha = 1.0$)	**no**	Not run—highest-priority gap (§10.2)
Multi-seed variance	**no**	Not run
Competitive baselines	**no**	Not run

SchemaForge v1.0 — August 2026. Author: Arjhine A. Ty. Production checkpoint: `schemaforge-1b-iter2`*. Distilled from* `google/gemma-4-31B` *and* `google/gemma-4-E4B-it` *into* `openbmb/MiniCPM5-1B` *on NVIDIA RTX PRO 6000 Blackwell Edition / Nebius AI Cloud.*