

BPCF: A Token-Efficient Process Model Notation for LLM-Based BPMN Generation

Ankit Agrawal, Nicholas Wolf, Flora Geske

SUMM AI GmbH

{ankit,nicholas,flora}@summ-ai.com

Abstract

Generating Business Process Model and Notation (BPMN) process models with LLMs is costly at the serialization layer. Standard XML and JSON formats consume hundreds to thousands of output tokens per model, creating severe latency and cost bottlenecks for interactive modeling assistants. We present BPCF (Business Process Compact Format), a compact, indentation-based notation designed specifically for LLM emission. BPCF-Full is lossless with respect to the structural constructs found in the PMo benchmark. At 221 tokens per model on average, BPCF-Full is roughly half the size of a Mermaid encoding, 8× smaller than verbose JSON representations (PME JSON) and about 29× smaller than BPMN XML. We evaluated nine BPMN representations across five LLMs and found that BPCF is the most token-efficient format by a wide margin, delivering the lowest cost and latency. Although exact accuracy rankings vary across models, BPCF offers a highly viable accuracy-efficiency trade-off for production systems, successfully modeling complex BPMN constructs where other compact formats fail. In the full-expressivity arm, BPCF-Full+Strict scores 0.469/0.472 on GPT-4o/Gemini at 148–155 visible output tokens. We publicly release our grammar, a dependency-free parser, and the full evaluation pipeline.

1 Introduction

LLM-powered process modeling assistants, which turn a natural-language description of a business process into an editable BPMN diagram (Von Rosing et al., 2015), have rapidly transitioned from research prototypes into commercial products. We operate one such platform in production, where every user interaction (including initial generation, conversational edits, and multi-turn regenerations) requires the LLM to emit a complete, updated process model. In this high-throughput environment,

the dominant driver of both API cost and user-facing latency is not the model’s internal reasoning, but its *serialization*: the sheer number of output tokens spent encoding the process graph.

Standard BPMN models are directed graphs containing *tasks* (units of work), *events* (start and end points), and *gateways* (decision and parallel splits, each later merged). Larger production models also introduce *pools* and *lanes* to assign organizational responsibilities, alongside cross-pool *message flows*. Compounding the serialization burden, the models LLMs emit are overwhelmingly *block-structured* (conditional splits have matching structural merges), so the generated topology is highly hierarchical and tree-like. Standard serialization formats fail to exploit this, forcing LLMs to explicitly define verbose, flat edge lists for structures that are fundamentally nested.

BPMN 2.0 XML (Dijkman et al., 2011; Von Rosing et al., 2015) represents the extreme case of this inefficiency: a medium-sized process costs several thousand output tokens (6,400 tokens on average in our benchmark), largely due to rigid layout coordinates and metadata with zero semantic value to the LLM. While practitioners use compact intermediate representations (IRs) and convert to XML post-inference (Köpke and Safan, 2024; Licardo et al., 2026), existing IRs remain sub-optimal. Structured JSON variants repeat keys and identifiers for every element. Meanwhile, highly compact alternatives like Mermaid or POWL achieve brevity only by sacrificing production-critical semantics, such as lanes, message flows, and typed tasks. Because no representation documented in comprehensive benchmarks (Brissard et al., 2025) is simultaneously compact, expressive, and accurate, industrial deployment remains financially and operationally constrained.

We present **BPCF** (Business Process Compact Format), a plain-text notation co-designed specifically with the structural traits of BPMN to mini-

mize tokens using four principles: (i) *Indentation* replaces explicit edge lists and merge gateways by capturing block-structured flows; (ii) *Lane inheritance* eliminates redundant organizational routing tags; (iii) *Identifier omission* drops unique IDs except on loopback jump targets; and (iv) *Type operators* compress node schemas into single characters. To balance parsing strictness with LLM flexibility, BPCF features three profiles: Strict, Core, and Full.

Our main contributions are as follows: (1) *The BPCF Formalism*: A grammar for three profiles and a dependency-free parser with automated repair mode for minor LLM syntax anomalies. (2) *PMo-Lossless Expressivity*: We demonstrate that BPCF-Full losslessly round-trips all 55 models of the PMo benchmark, establishing it as the only compact format capable of preserving all pools, lanes, and flows required by the dataset. (3) *Token Efficiency*: BPCF averages 221 tokens per model across the benchmark, an 87% reduction over optimized JSON and 50% over Mermaid. (4) *Empirical Evaluation*: We conduct a robust multi-run empirical evaluation of nine BPMN representations across five LLMs from three model families, finding that BPCF substantially cuts latency and cost while offering a practical accuracy-efficiency trade-off. (5) *Open Resources*: Grammar, parser, prompts, and pipeline are public.¹

2 Related Work

From Extraction to LLM Generation. The task of transforming natural language into business process models historically relied on hand-crafted rules to extract activities and control flow (Friedrich et al., 2011), highlighting deep linguistic bottlenecks (Van der Aa et al., 2017). The advent of LLMs largely overcame these extraction hurdles, making automated generation highly practical (Betencourt and Guerreiro, 2026; Lauer et al., 2026). However, this architectural shift introduced a new engineering bottleneck: serialization. While compiling the standard BPMN 2.0 XML was trivial for rule-based systems, it forces autoregressive LLMs to exhaust thousands of expensive output tokens on layout coordinates and boilerplate rather than focusing on semantic logic (Brissard et al., 2025; Licardo et al., 2026).

Intermediate Representations (IRs). To mitigate this token overhead, contemporary systems

typically task the LLM with generating an intermediate representation (IR) that is later compiled to XML. These approaches fall into three primary paradigms: (1) **Structured JSON IRs** mimic process trees and demonstrate massive token and latency reductions (Köpke and Safan, 2024; Licardo et al., 2026), but remain sub-optimal due to repetitive keys and element identifiers. (2) **Formal Grammars & Pipelines** generate algebraic expressions like POWL to guarantee soundness at the expense of modeling coverage (Kourani et al., 2024; Antonov et al., 2026), or rely on multi-stage execution repair and agentic orchestration (Matei et al., 2026; Lin et al., 2025). (3) **Compact Domain-Specific Languages (DSLs)**, such as Sketch Miner (Ivanchikj et al., 2022), have been adapted as LLM targets (Ziche and Apruzzese, 2024). However, because these were designed for human typing speed, they fail to exploit structural process redundancies systematically. Ultimately, as Brissard et al. (2025) demonstrated across nine formats on the PMo dataset, no single representation balances compactness, expressivity, and generation accuracy.

Token-Efficient Serialization & Positioning.

Beyond BPMN, optimizing data serialization is a first-class constraint in applied NLP (Xu et al., 2025), with domain-specific formats yielding massive efficiency gains (Dong et al., 2024). For domain-agnostic structured data, tabular formats like TOON (Schopplich et al., 2025) and JTON (Nandakishore, 2026) compress footprints by combining YAML-style nesting with CSV-style rows. Yet, independent benchmarks reveal these formats suffer from a steep “prompt tax” and struggle to generate deeply nested structures accurately in zero-shot settings (Matveev, 2026; Masciari et al., 2026). This limitation makes them particularly ill-suited for the deeply nested, tree-like gateway structures inherent to business process models. BPCF fills this distinct void: it is the first BPMN notation explicitly designed and benchmarked for LLM token efficiency. BPCF exploits domain properties like block structures and closed node types to compress models precisely where general formats fail. We evaluate this by benchmarking BPCF alongside a novel BPMN adaptation of TOON across multiple LLMs. This multi-model approach, combined with the representation-agnostic PME metric (Voelter et al., 2024), directly bridges the evaluation gaps left by prior studies.

¹<https://github.com/ankitagr01/bpcf-process-model-notation>

```

process OrderHandling
  start Order received
  T#check Check order details
  ? "Order complete?"
    "no":
      T Request missing information
      -> #check
    "yes":
      ||
      - T Reserve items
      - T Prepare invoice
      T Confirm order
  E Order confirmed

```

Figure 1: BPCF-Core: a process with an XOR gateway (?), a parallel block (||), a loop via anchor jump (-> #check), and an implicit merge before the final end event.

3 BPCF: Design and Notation

BPCF is a line-oriented, plain-text notation where each line represents a single process element. Structural hierarchy is natively expressed through indentation (two spaces per level), and free-text labels run to the end of the line. Figure 1 illustrates a complete process modeled in BPCF.

3.1 Design Principles

Four structural properties of process models drive BPCF’s token compression. **(P1) Block structure:** For block-structured regions, BPCF writes execution branches as indented blocks and leaves merges *implicit* (flow resumes at the gateway’s prior indentation level). This eliminates explicit join elements and verbose per-edge flow declarations. **(P2) Lane sparsity:** Elements share organizational lanes, so a lane index ([n]) is tagged only upon transition; subsequent elements inherit it. **(P3) Reference sparsity:** Because most elements are never referenced globally, unique identifiers are omitted. Only loopback or jump targets receive an anchor (e.g., T#check), which is referenced by a jump operator (-> #check), replacing the verbose id and reference mapping machinery. **(P4) Closed type system:** Finite BPMN node schemas are compressed into single characters: T for tasks (with suffixes u/s/m/>/< for user/service/manual/send/receive types), ? (XOR), || (AND), ?* (OR), ?= (event-based), E (end), and I(...) (intermediate events).

3.2 Profiles, Grammar, and Parser

The notation features three profiles to accommodate different generation requirements. **BPCF-Core** covers tasks, XOR/AND gateways, start/end

events, and jumps, matching the scope of prior text-to-process generation studies. **BPCF-Full** adds pools, lanes with inheritance, message flows (msg "label":: PoolA.#x -> PoolB.#y), typed tasks, and complex intermediate events and gateways. Conversely, **BPCF-Strict** applies rigorous structural discipline to the Core syntax, enforcing a constrained single-entry/single-exit structure: exactly one start and one end E event, no branch-level terminations, and backward-only loops. **BPCF-Full+Strict** applies the same Strict discipline to the Full syntax.

All profiles share a unified EBNF grammar (Appendix A) and a dependency-free reference parser. The parser supports a *strict* evaluator and a *tolerant* mode that automatically repairs minor, recoverable LLM syntax anomalies, such as irregular indentation. Because it logs every intervention, strict and repaired validity rates are reported independently (§5).

A compact notation must preserve semantic integrity to be viable. We validate BPCF via an exact round-trip evaluation: every PMo model is serialized to BPCF, parsed back, and compared against the original ground truth for element and flow parity. **BPCF-Full reproduces 55/55 models exactly**, fully preserving pools, lanes, message flows, typed tasks, and complex gateways. This semantic retention is unique among evaluated compact formats: JSON branches, BPMN Text, and POWL only express the 18 single-entry block-structured models, while Mermaid and Graphviz discard lane, type, and event semantics. The serializer recovers block structures via post-dominator analysis; any lossy conversion would surface as a structural mismatch during verification, and zero were detected across the benchmark suite.

4 Evaluation Setup

Dataset. We evaluate using the public PMo dataset (Brissard et al., 2025), which pairs 55 expert descriptions with BPMN ground truths. Our experimental design mirrors the established protocol across two configuration settings: a *basic* arm consisting of 45 models restricted to tasks, XOR/AND gateways, and start/end events, and a *full* arm containing all 55 models where pools, lanes, cross-pool message flows, and typed tasks are fully in scope.

Baselines. We implement five baseline formats directly from (Brissard et al., 2025): BPMN 2.0 XML, PME JSON (Voelter et al., 2024), JSON

branches (Köpke and Safan, 2024), BPMN Text (Lin et al., 2025), and Mermaid, utilizing the authors’ original prompt and scoring code for direct comparability. Additionally, we introduce and evaluate two distinct process model adaptations of the tabular TOON format (Schopplich et al., 2025): *TOON-flat* (a pure tabular implementation featuring separate element and edge tables) and *TOON-nested* (a YAML-style block structure layout).

Models. Generations are executed using GPT-4o (Hurst et al., 2024) and Gemini 2.5 Pro (128-token thinking budget) (Team et al., 2023). We additionally replicate the main comparison on three further models: Claude Sonnet 4.6 (Anthropic, 2026), GPT-5.1 (OpenAI, 2025), and Gemini 2.5 Flash (Team et al., 2023) (Appendix D). All evaluations are conducted at temperature 0 across three runs per cell. To account for non-deterministic API provider variance, we report mean scores paired with bootstrapped confidence intervals. The basic arm yields 2,430 generations (9 formats $\times$ 45 descriptions $\times$ 3 runs $\times$ 2 LLMs), while the full arm yields 1,980 generations (6 formats $\times$ 55 descriptions $\times$ 3 runs $\times$ 2 LLMs). BPCF-Strict was authored once as a separate instruction document rather than by appending rules to the Core prompt; BPCF-Full+Strict was likewise authored once and frozen before generation.

Prompts and Fairness of Comparison. To control semantic generation, all formats share an identical base prompt containing reference modeling guidelines, followed by a format-specific instruction block and a single execution example. The shared guidelines (361 tokens) are byte-identical across all nine formats, and every baseline instruction block is used verbatim from the PMo authors’ repository. BPCF’s format-specific block is the smallest (462 tokens vs. 964 for PME JSON and 2,708 for XML). BPCF’s own block underwent one revision during development, correcting an example that contradicted the shared guidelines (it taught branch-level end events); re-scoring the pre-revision generations bounds the revision’s effect at +0.005 (GPT-4o) and +0.014 (Gemini) PME. To verify that no conclusion depends on prompt engineering, we additionally evaluate BPCF under three instruction blocks authored from the grammar specification alone, with no benchmark access, frozen before use; all match or exceed the scores reported here (§5.4). We further test generalization on an independent, frozen out-of-domain dataset in

§5.5.

Metrics. We evaluate the experimental generations across three operational dimensions.

Accuracy is measured via Process Model Elements (PME) similarity (Voelter et al., 2024; Brisard et al., 2025): generated and reference elements are matched greedily using sentence-embedding cosine similarity (stsb-mpnet-base-v2, threshold 0.7) and scored with a frequency-aware Dice–Sørensen coefficient. The final metric is a union-weighted mean over task, event, and gateway names and types, plus flow triples. The primary unit of analysis is the description-level mean across three runs (we note a crucial single-reference caveat regarding these absolute scores in §5.2). We assess statistical separation with bootstrap 95% confidence intervals on paired per-description differences (Appendix Table 5), treating a difference as real only when its interval excludes zero; the full per-comparison bootstrap and Holm-corrected Wilcoxon outputs are released with the artifact. **Validity** tracks strict and repaired parse-compliance rates for BPCF and TOON, alongside schema-compilation success rates for baselines. **Efficiency** measures visible output tokens (re-tokenized with o200k_base), indicative wall-clock latency, and per-request dollar cost computed as total billed tokens at list prices. Static format size is evaluated independently by serializing the reference ground truths.

5 Results

5.1 Static Format Size

Before evaluating model generation, we first isolate format overhead by directly serializing the PMo ground truth (Appendix D). Under this static evaluation, BPCF-Full encodes all 55 models using a mean of just 221 tokens. This represents a 96.5% reduction compared to XML (6,384 tokens) and an 87% reduction compared to PME JSON (1,760 tokens). It is roughly half the size of Mermaid (433) and TOON-flat (420), while remaining the only compact format with zero information loss (§3.2). On the 18 block-structured models supported by the most compact prior formats, BPCF-Full requires 178 tokens, making it 44% smaller than POWL and 61% smaller than JSON branches. Ablating the four design principles on this static serialization (Table 12) shows compression is driven primarily by identifier omission (P3, +31–34% when disabled) and single-character types (P4), while lane

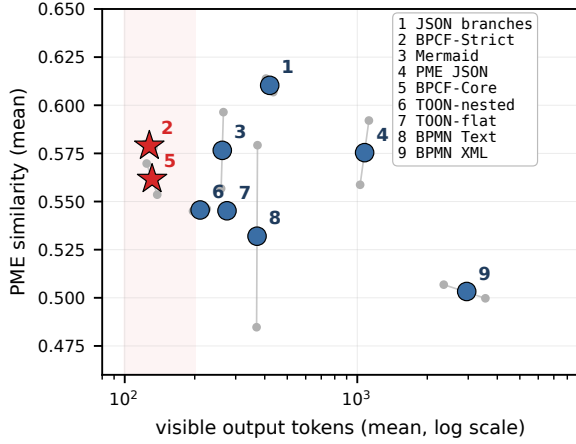


Figure 2: Accuracy vs. output size, basic arm. Red stars are BPCF profiles; blue circles the baselines (per-format means over both models); grey dots are individual LLM models. Token counts use the o200k_base tokenizer.

inheritance contributes only on the 6 multi-pool models and implicit merges pay off only in generated output, whose topology, unlike the references’, is block-structured.

5.2 Zero-Shot Generation (Basic Arm)

Figure 2 summarizes the 45-model generation experiment; Table 1 provides the full numbers. We highlight three main findings:

Accuracy and Error Analysis. Differences in PME scores must be interpreted carefully due to the metric’s single-reference limitation (§4). A small gap often reflects closeness to a single human-authored reference rather than a true quality difference. We report accuracy conservatively. While Table 1 uses BPCF’s deployed tolerant parser, our baseline comparisons strictly zero out any unparseable outputs to ensure fairness. Under this strict scoring, BPCF-Strict achieves 0.553 on GPT-4o and 0.558 on Gemini, while BPCF-Core scores 0.498 and 0.562. JSON branches remains the most accurate format overall (0.607 and 0.614), leading BPCF-Strict by roughly 0.05. This gap is statistically significant. However, BPCF-Strict performs comparably to PME JSON and Mermaid on GPT-4o, though it trails them slightly on Gemini. It consistently matches or exceeds the TOON variants and XML (Appendix Table 5).

The Strict profile significantly improves upon Core (e.g., +0.055 on GPT-4o under strict scoring) primarily by suppressing spurious end events: Core emits 1.47 end events per model against a reference mean of 0.98, and Strict reduces this

to 1.0 while forbidding early branch terminations and forward jumps, acting as a guardrail against LLM routing errors. Notably, the gain does not come from replicating the references’ authoring shape: only 1 of 45 references is itself a single-entry, single-exit tree (43 of 45 place an end event inside a branch). Strict instead matches the end-event *count*, which the metric scores, at the cost of a genuine expressivity restriction (§6). This discipline improves both syntax validity and gateway sub-scores (Appendix D), and single-reference metrics inherently reward such agreement with the one reference model.

Output size and validity. BPCF generates the most compact outputs across all nine formats, averaging 122 to 138 visible tokens. In comparison, Mermaid requires roughly 260 tokens, JSON branches 420, and PME JSON over 1,000 (Table 1). Despite being absent from the models’ training data, BPCF achieves a strict parse rate of 90% to 99%. While our tolerant parser can recover most remaining errors (Appendix Table 8), we rely on strict parsing for conservative evaluation to avoid unfairly favoring BPCF. Conversely, formats like TOON-nested struggle with validity; its YAML-style syntax breaks when labels contain colons, dropping its strict parse rate to 68% on GPT-4o. BPCF’s line-based grammar completely avoids this failure mode.

5.3 Full-Expressivity Arm

Our second experiment evaluates all 55 descriptions using the full PME profile, introducing pools, lanes, message flows, and typed tasks. PME JSON receives extended instructions for these elements, whereas Mermaid natively lacks support for them.

This setting proves harder, with scores dropping by roughly 0.10 across all formats (see Appendix Table 2). The primary difficulty lies in determining whether a description warrants pools, a feature used in only 6 of the 55 reference models. JSON branches leads both models at roughly 0.51, outperforming BPCF-Full by 0.08. However, JSON branches achieves this by omitting pools, lanes, and message flows entirely. Among formats that actually support these elements, BPCF-Full scores 0.441. On GPT-4o, BPCF-Full is comparable to PME JSON, Mermaid, and TOON-flat (within 0.02 PME), though it clearly trails the leaders on Gemini. Combining Full expressivity with Strict discipline closes this gap. Because Strict is a prompt-level dis-

Model	Format	PME $\uparrow$	95% CI	Strict val.	Vis. tokens $\downarrow$	Latency $\downarrow$	\$/proc. $\downarrow$
GPT-4o	JSON branches	0.607	[.563, .650]	1.00	435	4.7 s	.007
GPT-4o	BPMN Text	0.579	[.525, .632]	0.96	372	4.9 s	.007
GPT-4o	BPCF-Strict	0.578 (.553)	[.539, .617]	0.96	133	1.9 s	.004
GPT-4o	PME JSON	0.559	[.514, .604]	1.00	1,029	8.1 s	.014
GPT-4o	Mermaid	0.557	[.510, .603]	1.00	260	3.9 s	.005
GPT-4o	BPCF-Core	0.554 (.498)	[.516, .591]	0.90	138	3.4 s	.004
GPT-4o	TOON-nested	0.546	[.511, .582]	0.68	225	3.8 s	.005
GPT-4o	TOON-flat	0.544	[.503, .584]	0.94	279	4.4 s	.006
GPT-4o	BPMN XML	0.507	[.450, .560]	0.92	2,354	20.7 s	.032
Gemini 2.5 Pro	JSON branches	0.614	[.570, .657]	1.00	405	6.3 s	.007
Gemini 2.5 Pro	Mermaid	0.596	[.553, .643]	1.00	266	4.5 s	.005
Gemini 2.5 Pro	PME JSON	0.592	[.550, .632]	1.00	1,122	12.2 s	.016
Gemini 2.5 Pro	BPCF-Strict	0.580 (.558)	[.541, .619]	0.97	122	4.5 s	.003
Gemini 2.5 Pro	BPCF-Core	0.570 (.562)	[.530, .610]	0.99	124	4.2 s	.003
Gemini 2.5 Pro	TOON-flat	0.547	[.513, .580]	0.92	272	5.2 s	.005
Gemini 2.5 Pro	TOON-nested	0.545	[.504, .588]	0.93	197	4.5 s	.004
Gemini 2.5 Pro	BPMN XML	0.500	[.439, .561]	0.91	3,555	33.2 s	.047
Gemini 2.5 Pro	BPMN Text	0.485	[.456, .515]	1.00	369	5.6 s	.006

Table 1: Zero-shot generation on the PMo basic subset (45 descriptions $\times$ 3 runs). Main PME scores use the tolerant parser; values in parentheses are strict-only scores (non-strict outputs zeroed). \$/proc. is the total billed cost at list prices.

cipline over the unchanged grammar, it composes freely with pools and lanes: BPCF-Full+Strict scores 0.469 (GPT-4o) and 0.472 (Gemini), significantly above plain BPCF-Full (+0.029/+0.035, bootstrap CIs exclude zero) and statistically inseparable from the full-arm leaders PME JSON (0.462/0.477) and Mermaid (0.461/0.487), at 148–155 output tokens. Notably, BPCF-Full remains the only format that can represent all PMo constructs without data loss, while using up to 7 times fewer tokens. The fact that PME JSON’s accuracy drops when provided with pool instructions highlights a broader calibration challenge shared by all expressive formats. In an industry setting, expressivity is a practical requirement rather than a benchmark optimization. Enterprise users rely on lanes and message flows to map real-world organizational responsibilities, making support for these elements necessary in production software despite the scoring penalty.

5.4 Robustness Checks

To verify stability, we tested two paraphrased versions of the modeling instructions (Appendix Table 6). Rewording shifts accuracy by up to 0.055 for Mermaid and 0.021 for BPCF, indicating that small ranking differences among top formats are highly sensitive to prompt phrasing. Finally, our core findings replicate successfully on a smaller model (Gemini 2.5 Flash; Appendix Table 9): the Strict profile consistently outperforms Core, and

no baseline beats BPCF-Strict by more than a 0.05 margin.

Prompt-source robustness. To rule out a prompt-engineering advantage, we had three BPCF instruction blocks authored from scratch by isolated LLM instances given only the published grammar and the shared modeling guidelines, with no access to PMo, to any generation or score, or to our own prompts; each block was frozen before use and never revised. Across 3 blocks $\times$ 2 models $\times$ both profiles, 11 of 12 comparisons match or exceed the hand-written block used elsewhere in this paper (the single exception is -0.001). Untuned BPCF-Strict reaches 0.593–0.629 on GPT-4o (vs. 0.578 reported) and 0.597–0.609 on Gemini (vs. 0.580). The pattern extends to the added model families: with an untuned block, BPCF-Strict is statistically inseparable from JSON branches on Claude Sonnet 4.6 (0.633 vs. 0.635) at $2.9\times$ fewer output tokens, and ranks second of five formats on GPT-5.1 (0.596, behind PME JSON at 0.603). BPCF’s competitiveness therefore does not depend on prompt tuning; if anything, our primary tables are conservative. Because baselines retain their original prompts, this establishes that BPCF requires no tuning to be competitive, not that it wins under equal prompt-engineering effort. Independently authored blocks span 0.011–0.037 PME, so instruction wording matters for every format; per-block numbers are in Appendix Table 11.

5.5 Held-Out Replication

To evaluate generalization beyond PMo, we tested BPCF on 20 held-out process descriptions from the PET dataset (Bellan et al., 2022), alongside JSON branches and PME JSON, with all configurations frozen. BPCF matched its PMo token reduction and strict validity (90% to 95%). Its task accuracy trails PME JSON by a wider margin here than on PMo (0.47 vs. 0.57 on GPT-4o; Appendix C), but the loss is concentrated rather than uniform: 5 of the 20 documents carry 70% of it, and these are the shorter ones (the gap shrinks with document length, $r = -0.43$). Because PET’s gold annotations contain no gateway or event structure, our approximate reference covers only tasks and flows; within that scope, the loss traces to over-compression on short descriptions and to label granularity (PET annotates bare verb phrases, whereas BPCF emits actor-prefixed labels that embedding matching scores as misses). We therefore treat PET as evidence that validity and token efficiency transfer to unseen data, and regard the cross-format accuracy comparison as indicative only, since flow accuracy was low for all formats under this approximate reference.

5.6 Cost in Production Terms

Since models generate output tokens serially, format size dictates latency: BPCF completes generation in 1.9 s (95% CI [1.7, 2.2]) to 4.5 s [4.1, 5.0], compared to 8.1 s [7.3, 9.0] to 12.2 s [10.8, 13.7] for PME JSON and up to 33.2 s [29.7, 36.7] for XML; Gemini 2.5 pro latencies additionally include a fixed 128-token thinking budget that affects latency independently of format size. Billing shows a similar advantage. Scaling GPT-4o to 10,000 generations per day, BPCF costs approximately \$40, whereas PME JSON reaches \$140 and XML \$320.²

6 Discussion and Outlook

Why formats differ: structural discipline, not syntax. Task sub-scores are nearly identical across the top formats (0.67–0.71; Appendix Table 4); accuracy differences come almost entirely from control flow: gateways, merges, and end events. Formats whose syntax enforces structure do better because several common modeling errors

become impossible to write: JSON branches mandates a single start, a single end, and a block structure, and BPCF-Strict isolates exactly this effect, reducing spurious end events from 1.47 to 1.0 per model and closing most of the gap. Notably, the references themselves are not single-exit trees (§5.2), so what discipline buys is agreement on element counts, which the metric scores, rather than stylistic imitation. Verbosity is a second factor: the most verbose format, BPMN XML, is consistently the least accurate. Pre-training familiarity matters less than one might expect: BPCF is absent from training data yet competitive, and blocks authored blind from its grammar perform at least as well as our hand-written one (§5.4). The full-expressivity arm shows the reverse face of discipline: given freedom to introduce pools and lanes, models frequently misjudge their necessity and accuracy drops. Structural freedom adds ways to fail; discipline removes them. BPCF therefore offers adjustable profiles so systems can tune this freedom without inflating token costs.

Outlook. Three directions remain for future work: (1) *Conversational editing*. Evaluating conversational editing, where compounding prompt costs make token-efficient formats a practical necessity, but BPCF’s lack of stable node IDs must be tested against ID-based formats during targeted updates; (2) *Grammar-constrained decoding* on open-weight models would guarantee validity, though our data suggests it would not buy accuracy (generations requiring repair score no worse than strictly valid ones), and BPCF’s indentation-sensitivity requires a depth-bounded grammar for CFG-based samplers. (3) *Input preprocessing*. Most production failures trace to vague descriptions (implicit decisions, unclear actors, unstated parallelism); an LLM normalization step before generation is a promising lever. We release our grammar, parser, prompts, and evaluation pipeline to support industry adoption.

Limitations

Our evaluation is bounded in several ways.

- **Dataset overlap.** While PMo is highly diverse, our notation and parser were developed against this dataset. Our untuned-prompt control (§5.4) and the held-out PET test (§5.5) bound this risk for the prompt and for validity respectively, but a full replication with

²Computed from generated (not static) token counts: mean billed input plus output tokens per request at GPT-4o list prices (\$2.50/M input, \$10/M output). E.g. BPCF-Strict: $(1,094 \times 2.5 + 134 \times 10)/10^6 \approx \$0.004$ per process $\times 10,000$.

aligned references is still needed. Our findings are also limited to zero-shot, English-only prompts.

- **Partial BPMN 2.0 coverage.** BPCF is loss-less with respect to the PMo benchmark, but it deliberately omits constructs absent from that dataset, such as boundary events, subprocesses, and data objects.
- **Metric constraints.** PME similarity measures structural correspondence against a single human reference rather than true behavioral equivalence. Consequently, perfectly valid alternative process models are penalized, meaning all absolute scores must be interpreted comparatively.
- **No conversational-editing evaluation.** Our experiments test initial process generation rather than multi-turn editing. Because BPCF omits stable node IDs, edits require re-emitting enclosing blocks rather than patching specific nodes. Whether this approach matches the editing accuracy of ID-based formats remains an open question.
- **Zero training data handicap.** Unlike XML or JSON, BPCF is entirely absent from LLM pre-training. Our results rely entirely on in-prompt grammar definitions, which explains why raw validity sits slightly below 100%. To prevent our tolerant parser from masking these errors by dropping unparseable lines, we consistently report strict-only scores alongside them.
- **Prompt provenance.** BPCF’s instruction block received one revision during development (bounded at $+0.005/+0.014$ PME, §4), and the untuned-prompt control authors fresh blocks only for BPCF while baselines keep their original prompts; it therefore shows BPCF needs no tuning to be competitive, not that it wins at equal prompt-engineering effort. Independently authored instruction blocks shift BPCF by $0.011\text{--}0.037$ PME, so instruction wording is a meaningful variance source for every format. Latency measurements reflect shared cloud endpoints and are indicative.

Ethical Considerations

This work relies exclusively on the public, expert-curated PMo benchmark, hosted on Zenodo under a Creative Commons license. The research

involves no human subjects, personal data, or sensitive content. We utilize the benchmark authors’ baseline conversion code strictly for evaluation. Because this external code lacks an open-source license, we do not redistribute it; our released artifacts contain only original code and reference the upstream repository. BPCF is designed to reduce the compute cost and user latency of LLM-based process modeling tools. We foresee no harmful applications of this work beyond the standard socioeconomic impacts generally associated with business process automation.

References

- Anthropic. 2026. Claude Sonnet 4.6 system card. <https://www.anthropic.com/claude-sonnet-4-6-system-card>. Model version claude-sonnet-4-6, accessed via Vertex AI.
- Anton Antonov, Humam Kourani, Alessandro Berti, and Gyunam Park. 2026. Beyond control-flow: Integrating the resource perspective into multi-collaborative process modeling from text. *arXiv preprint arXiv:2605.24546*.
- Patrizio Bellan, Han van der Aa, Mauro Dragoni, Chiara Ghidini, and Simone Paolo Ponzetto. 2022. Pet: an annotated dataset for process extraction from natural language text tasks. In *International Conference on Business Process Management*, pages 315–321. Springer.
- João Bettencourt and Sérgio Guerreiro. 2026. Large language models to enhance business process modeling: Past, present, and future trends. *arXiv preprint arXiv:2604.14034*.
- Alexis Brissard, Frédéric Cuppens, and Amal Zouaq. 2025. What is the best process model representation? a comparative analysis for process modeling with large language models. In *International Conference on Business Process Management*, pages 55–68. Springer.
- Remco Dijkman, Jorg Hofstetter, and Jana Koehler. 2011. *Business process model and notation*, volume 89. Springer.
- Haoyu Dong, Jianbo Zhao, Yuzhang Tian, Junyu Xiong, Mengyu Zhou, Yun Lin, José Cambronero, Yeye He, Shi Han, and Dongmei Zhang. 2024. Encoding spreadsheets for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 20728–20748.
- Fabian Friedrich, Jan Mendling, and Frank Puhlmann. 2011. Process model generation from natural language text. In *International conference on advanced information systems engineering*, pages 482–496. Springer.

- Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.
- Ana Ivanchikj, Souhaila Serbout, and Cesare Pautasso. 2022. Live process modeling with the bpmn sketch miner: A. ivanchikj et al. *Software and systems modeling*, 21(5):1877–1906.
- Julius Köpke and Aya Safan. 2024. Introducing the bpmn-chatbot for efficient llm-based process modeling. In *BPM (Demos/Resources Forum)*, pages 86–90.
- Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil MP van der Aalst. 2024. Process modeling with large language models. In *International Conference on Business Process Modeling, Development and Support*, pages 229–244. Springer.
- Chantale Lauer, Peter Pfeiffer, Alexander Rombach, and Nijat Mehdiyev. 2026. Assessing the business process modeling competences of large language models. *arXiv preprint arXiv:2601.21787*.
- Josip Tomo Licardo, Nikola Tanković, and Darko Etinger. 2026. Bpmn assistant: An llm-based approach to business process modeling. *Applied Sciences*, 16(5):2213.
- Leilei Lin, Yumeng Jin, Yingming Zhou, Wenlong Chen, and Chen Qian. 2025. Mao: A framework for process model generation with multi-agent orchestration. *IEEE Transactions on Services Computing*.
- Elio Masciari, Vincenzo Moscato, Enea Vincenzo Napolitano, Gian Marco Orlando, Marco Perillo, and Diego Russo. 2026. Are llms ready for toon? benchmarking structural correctness-sustainability trade-offs in novel structured output formats. *arXiv preprint arXiv:2601.12014*.
- Ion Matei, Maksym Zhenirovskyy, Praveen Kumar Menaka Sekar, and Hon Yung Wong. 2026. Automated bpmn model generation from textual process descriptions: A multi-stage llm-driven approach. In *2026 IEEE International Systems Conference (SysCon)*, pages 1–7. IEEE.
- Ivan Matveev. 2026. Token-oriented object notation vs json: A benchmark of plain and constrained decoding generation. *arXiv preprint arXiv:2603.03306*.
- Gowthamkumar Nandakishore. 2026. Jton: A token-efficient json superset with zen grid tabular encoding for large language models. *arXiv preprint arXiv:2604.05865*.
- OpenAI. 2025. GPT-5.1 Instant and GPT-5.1 Thinking system card addendum. <https://openai.com/index/gpt-5-system-card-addendum-gpt-5-1/>. Model version gpt-5.1-2025-11-13, accessed via Azure OpenAI.
- Johann Schopplich and 1 others. 2025. TOON: Token-oriented object notation. <https://github.com/toon-format/toon>. Specification v3.3, github.com/toon-format/spec.
- Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.
- Han Van der Aa, Henrik Leopold, and Hajo A Reijers. 2017. Comparing textual descriptions to process models—the automatic detection of inconsistencies. *Information Systems*, 64:447–460.
- Marvin Voelter, Raheleh Hadian, Timotheus Kampik, Marius Breitmayer, and Manfred Reichert. 2024. Leveraging generative ai for extracting process models from multimodal documents. *arXiv preprint arXiv:2406.04959*.
- Mark Von Rosing, Stephen White, Fred Cummins, and Henk De Man. 2015. Business process model and notation-bpmn.
- Zhentao Xu, Fengyi Li, Albert C Chen, and Xiaofeng Wang. 2025. Procuto: Llm prompt compression via attribution estimation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 285–309.
- Clara Ziche and Giovanni Apruzzese. 2024. Llm4pm: A case study on using large language models for process modeling in enterprise organizations. In *International conference on business process management*, pages 472–483. Springer.

A BPCF by Example

This appendix demonstrates how BPCF represents each BPMN construct and concludes with the Core grammar. We use two-space indentation to encode nesting, and labels run to the end of the line.

Tasks and events. A task occupies one line beginning with T. An optional type suffix marks user, service, manual, send, or receive tasks using Tu, Ts, Tm, T>, and T< respectively. The process opens with start and a branch ends with E. Intermediate catching events use I(msg), I(timer), or I(cond).

Exclusive gateway (decision). A ? line opens a decision; each alternative is an indented "condition": line whose body is nested one level deeper. After the block, flow rejoins automatically:

```
? "Order complete?"
  "no":
    T Request missing information
  "yes":
    T Reserve items
T Confirm order      <- both paths reach here
```

```
||
- T Reserve items
- T Prepare invoice
T Confirm order      <- after the AND-join
```

```
T#review Review request
? "Approved?"
  "no":
    T Revise
    -> #review      <- loop back
  "yes":
    T Archive
```

```
pool Order
  lanes: Sales, Warehouse
  start Order received
  T Record order          <- lane 1 (Sales)
  T[2] Pick items         <- switches to Warehouse
  T#ship Ship order       <- still Warehouse
  E Done
msg "shipment" :: Order.#ship -> Carrier
```

```

process    ::= "process" SP name NL
            INDENT start seq DEDENT
start      ::= "start" (SP label)? NL
seq        ::= task+
node       ::= node | gateway | jump | end
task       ::= "T" ttype? anchor? SP label NL
ttype      ::= "u" | "s" | "m" | ">" | "<"
gateway    ::= gsym anchor? (SP gname)? NL
            (INDENT branch+ DEDENT)?
gsym       ::= "?" | "|" | " " | "*" | "=" | "?" | "c"
branch     ::= (qstring | "-" ) ":"?
            (SP inline)? NL
            (INDENT seq DEDENT)?
inline     ::= task | gateway | jump | end
jump       ::= "->" SP "#" ident NL
end        ::= "E" anchor? (SP label)? NL
anchor     ::= "#" ident

```

implied when two or more merge; (3) a single-branch gateway is an if-then whose implied else falls through; (4) a gateway with no branch block is an explicit join marker (loop heads); (5) `-> #x` jumps backward (loops) or forward (cross-branch merges); (6) E terminates its branch. When the inline node on a branch line is itself a gateway, the gateway keyword sits inline and its branch block opens on the following, more-indented lines (a single line never contains a whole gateway block). The Strict profile additionally forbids branch-level E and forward jumps and requires a single trailing E. The Full profile adds pools, lane inheritance, typed start events, intermediate events, the `?*/?=//?c` gateways, and message flows. The machine-readable EBNF for all profiles, the dependency-free parser, and every BPCF formatting-instruction block ship with the artifacts; baseline blocks are obtained from the upstream PMo repository.

```
process: Order Handling
elements[4]{id,type,name}:
  n1,start,Order received
  n2,task,Check order details
  n3,xor,Order complete?
  n4,end,Order confirmed
flows[3]{from,to,condition}:
  n1,n2,
  n2,n3,
  n3,n4,yes
```

This appendix details the held-out replication summarized in §5.5. Because the notation, prompt, and parser were developed exclusively on PMo, we must verify whether BPCF’s behavior transfers to external data rather than merely adapting to our training set. We utilize PET (Bellan et al., 2022), an independent process-extraction dataset. We select its 20 longest documents that do not appear in PMo, ensuring the text is both unseen and

Format	PME $\uparrow$		Succ.	Tokens
	4o	Gem		
JSON branches	.518	.514	.96–1.0	435
Mermaid	.461	.487	.98–.99	285
PME JSON	.462	.477	1.00	1,111
BPMN Text	.483	.433	.92–.93	392
TOON-flat	.456	.449	1.00	310
BPCF-Full	.441	.438	1.00	158
BPCF-Full+Strict	.469	.472	1.00	151

Table 2: Full-expressivity arm results (§5.3) for all 55 descriptions using identical scoring with pools, lanes, and message flows in scope. Succ. represents the share of outputs convertible to a scoreable model, where failures count as zero in the mean. Tokens indicate the mean visible output tokens across both models.

comparable in complexity. We regenerate these models with the final prompts and parser frozen across BPCF-Strict, BPCF-Core, JSON branches, and PME JSON. We omit Mermaid and TOON as they add little comparative value here.

For reference-free metrics, we calculate strict-parse validity as the fraction of outputs parsing successfully under the strict grammar. This metric is meaningful only for BPCF since the baselines are scored by their own native converters. Output size is measured in visible tokens.

To estimate accuracy, we use the human gold annotations provided with PET (Bellan et al., 2022). We convert these annotations into approximate references featuring one task per activity and one flow per gold relation. We score tasks and flows using the standard PME matching algorithm. Because PET lacks gateway, event, or condition structures in PME form and uses a different activity granularity than PMo, absolute values are naturally lower and noisier. Consequently, only the relative comparison across formats is meaningful. A task-similarity of 0.5 indicates that roughly half of the reference activities matched successfully. The primary take-away remains comparative: BPCF performs within the baseline range for tasks, and all formats score predictably low on flows.

D Additional Results

The following tables support the claims made in §5 and derive directly from the released result files. Table 3 provides the complete static token analysis across all formats and coverage panels. Table 4 breaks the basic-arm scores into PME sub-scores, confirming that the Strict profile’s accuracy gains over Core occur primarily in gateway and flow cate-

Format	Mean	Median
<i>basic-45</i> (n=45 models)		
BPCF-Core	210	194
Mermaid (basic)	378	349
TOON-flat (basic)	405	381
Graphviz (basic)	578	549
Simplified XML (basic)	1,230	1,141
PME JSON (basic)	1,702	1,583
BPMN Process (basic)	1,915	1,783
<i>block-18</i> (n=18 models)		
BPCF-Full	178	145
POWL	319	272
BPMN Text	392	340
JSON branches	451	391
<i>full-55</i> (n=55 models)		
BPCF-Full	221	210
TOON-flat	420	406
Mermaid	433	410
Graphviz	639	602
Simplified XML	1,418	1,178
PME JSON	1,760	1,604
BPMN Process	1,990	1,855
BPMN 2.0 XML	6,384	5,955

Table 3: Static token counts (o200k_base) of ground-truth serializations across all evaluation subsets. The “basic” subset uses the simplified 45-model ground truth, while the “block-18” subset contains only models expressible by the most restricted formats.

gories rather than tasks. Table 6 details the prompt-paraphrase study. Table 9 presents the Gemini 2.5 Flash small-model replication. Table 7 provides the full per-format and per-model breakdown of the PET held-out replication. Finally, Table 8 lists all repairs made by the tolerant parser across the BPCF-Core generations. The largest repair category involves skipping unparseable lines, which can drop elements entirely. This behaviour underscores why we consistently report strictly valid scores alongside the tolerant results §5.2. Table 10 reports the Claude Sonnet 4.6 and GPT-5.1 replications, Table 11 the per-block untuned-prompt results, and Table 12 the design-principle ablation. Of the 20 skipped lines in Table 8, occurring in 15 of 270 generations, 8 are XOR branch-condition lines (the harmful class: dropping a branch removes a gateway path, which is why damage concentrates in gateway and flow sub-scores), 8 are anchor misuse, 2 are jumps to invalid targets, and 2 are other; since a skipped line deletes an element, tolerant parsing lowers rather than inflates BPCF’s score.

Format	Overall	Tasks	Gateways	Flows
<i>gemini-2.5-pro</i>				
JSON branches	0.614	0.700	0.604	0.533
Mermaid	0.596	0.709	0.502	0.474
PME JSON	0.592	0.682	0.558	0.491
BPCF-Strict	0.580	0.687	0.560	0.472
BPCF-Core	0.570	0.699	0.526	0.456
TOON-flat	0.547	0.643	0.519	0.453
TOON-nested	0.545	0.668	0.514	0.428
BPMN 2.0 XML	0.500	0.586	0.457	0.407
BPMN Text	0.485	0.510	0.607	0.355
<i>gpt-4o</i>				
JSON branches	0.607	0.700	0.598	0.520
BPMN Text	0.579	0.663	0.560	0.469
BPCF-Strict	0.578	0.679	0.560	0.476
PME JSON	0.559	0.669	0.498	0.430
Mermaid	0.557	0.702	0.399	0.410
BPCF-Core	0.554	0.673	0.514	0.442
TOON-nested	0.546	0.675	0.494	0.432
TOON-flat	0.544	0.697	0.446	0.416
BPMN 2.0 XML	0.507	0.610	0.443	0.407

Table 4: PME sub-scores for the basic arm (45 descriptions $\times$ 3 runs). Tasks and Gateways combine name and type matching, while Flows represent source-condition-target triples.

Baseline	Δ PME	95% CI
<i>gpt-4o</i>		
JSON branches	-0.054	[-0.102, -0.015]*
PME JSON	-0.005	[-0.052, +0.034]
Mermaid	-0.004	[-0.053, +0.039]
BPMN Text	-0.026	[-0.082, +0.025]
TOON-flat	+0.010	[-0.032, +0.046]
TOON-nested	+0.007	[-0.034, +0.041]
BPMN 2.0 XML	+0.046	[-0.011, +0.100]
<i>gemini-2.5-pro</i>		
JSON branches	-0.056	[-0.096, -0.019]*
PME JSON	-0.034	[-0.070, -0.001]*
Mermaid	-0.038	[-0.079, -0.000]*
BPMN Text	+0.074	[+0.041, +0.107]*
TOON-flat	+0.012	[-0.022, +0.041]
TOON-nested	+0.013	[-0.025, +0.047]
BPMN 2.0 XML	+0.059	[+0.000, +0.126]*

Table 5: Paired per-description difference (BPCF-Strict minus each baseline) under strict-only scoring, where non-strict BPCF outputs are zeroed. Positive values favor BPCF-Strict. An asterisk (*) indicates a bootstrap confidence interval that excludes zero. JSON branches is the only baseline significantly ahead of BPCF-Strict on both models.

Format	Model	Orig.	Para. A	Para. B	Spread
BPCF-Core	gemini-2.5-pro	0.578	0.579	0.575	0.004
BPCF-Core	gpt-4o	0.558	0.547	0.536	0.021
PME JSON	gemini-2.5-pro	0.608	0.614	0.627	0.019
PME JSON	gpt-4o	0.578	0.571	0.579	0.008
Mermaid	gemini-2.5-pro	0.611	0.618	0.563	0.055
Mermaid	gpt-4o	0.588	0.585	0.597	0.012

Table 6: Prompt-wording sensitivity. Shows PME similarity using the original shared modeling instructions versus two paraphrased versions on a 10-description subset. Spread indicates the maximum minus the minimum score.

Format	Strict val.	Tokens	Task sim	Flow sim
<i>gpt-4o</i>				
BPCF-Strict	95%	197	0.473	0.304
BPCF-Core	90%	199	0.456	0.278
JSON branches	—	550	0.447	0.254
PME JSON	—	1323	0.575	0.343
<i>gemini-2.5-pro</i>				
BPCF-Strict	90%	181	0.493	0.274
BPCF-Core	90%	183	0.511	0.305
JSON branches	—	557	0.561	0.297
PME JSON	—	1342	0.523	0.306

Table 7: Post-freeze held-out replication on 20 unseen PET descriptions. Strict validity and tokens are reference-free metrics. Task and flow similarity use approximate references derived from PET’s independent gold annotations. Because absolute values here are naturally noisier than on PMo, only relative comparisons between formats are meaningful.

Deviation repaired by the tolerant parser	Count
unrecoverable line (skipped)	20
indentation level off by one	12

Table 8: Repair taxonomy across all 270 BPCF-Core generations (45 basic-arm models $\times$ 3 runs $\times$ 2 models). A total of 255 generations parsed strictly with no intervention. Because skipping an unparseable line can drop an element and artificially lower the PME score, tolerant parsing does not uniformly inflate accuracy.

Format	PME	Strict val.	Tokens	Latency
JSON branches	0.603	1.00	432	2.7 s
BPMN Text	0.590	0.98	384	2.3 s
PME JSON	0.579	1.00	1174	6.1 s
Mermaid	0.566	0.98	266	1.7 s
TOON-flat	0.564	0.84	291	2.1 s
BPCF-Strict	0.563	0.91	143	1.2 s
BPCF-Core	0.533	0.98	123	1.1 s
TOON-nested	0.529	0.87	221	1.6 s
BPMN 2.0 XML	0.476	0.87	3763	18.0 s

Table 9: Small-model replication using Gemini 2.5 Flash with thinking disabled. The accuracy gain of the Strict profile over Core replicates successfully (+0.030). BPCF-Strict remains the smallest and fastest format while performing competitively against top baselines.

Format	PME $\uparrow$	Strict val.	Vis. tok. $\downarrow$
<i>Claude Sonnet 4.6</i>			
JSON branches	0.635	1.00	536
BPCF-Core (untuned)	0.634	0.92	191
BPCF-Strict (untuned)	0.633	0.93	186
Mermaid	0.627	0.99	321
PME JSON	0.621	0.97	1,464
BPCF-Strict	0.590	0.89	169
BPCF-Core	0.583	0.93	168
<i>GPT-5.1</i>			
PME JSON	0.603	1.00	1,348
BPCF-Strict (untuned)	0.596	0.99	202
BPCF-Strict	0.584	0.96	182
BPCF-Core (untuned)	0.581	0.96	215
JSON branches	0.580	0.98	557
Mermaid	0.578	1.00	352
BPCF-Core	0.558	0.93	185

Table 10: Replication on two model families added after submission (45 scored descriptions $\times$ 3 runs). “(untuned)” rows use instruction blocks authored from the grammar alone (§5.4).

Profile	Model	Paper	B1	B2	B3
Core	GPT-4o	0.554	0.617	0.600	0.584
Core	Gemini 2.5 Pro	0.570	0.605	0.568	0.591
Strict	GPT-4o	0.578	0.629	0.593	0.613
Strict	Gemini 2.5 Pro	0.580	0.597	0.602	0.609

Table 11: Untuned-prompt control: PME of three independently authored, benchmark-blind BPCF instruction blocks (B1–B3) against the hand-written block used in the main tables (“Paper”).

Principle disabled	basic-45	full-55
P3: identifier omission	+34%	+31%
P4: single-char types	+7.5%	+12%
P2: lane inheritance	0%	+1.4%
P1: implicit merge	0%	0%
All four	+40%	+41%

Table 12: Static-serialization token overhead when each design principle is disabled. P1 shows no static effect because the reference graphs are largely non-block-structured (none of their 182 joins can be absorbed); its saving materializes in generated output, which is block-structured (0.5 vs. 5.5 jumps per model).