GIM: Guided Infilling Modeling for Structured Knowledge Mining from LLMs

Anonymous ACL submission

Abstract

Utilizing large language models (LLMs) for industrial-grade data mining is hindered by a fundamental **impedance mismatch**: the probabilistic, unstructured nature of LLM generation versus the rigid structural requirements of knowledge discovery pipelines. Current alignment methods fall short: constrained decoding techniques typically decouple schema constraints from the semantic context, leading to suboptimal alignment. We introduce **Guided Infilling Modeling (GIM)**, a novel paradigm that reframes structured knowledge mining from LLMs as a guided, constrained infilling task. At its core, GIM compiles structural constraints directly into the input context, and employs constrained decoding at the output stage to achieve joint semantic and structural guidance, alleviating the impedance mismatch.

Extensive experiments demonstrate that: (1) **Synergy of Structure and Semantics:** GIM resolves the trade-off between structural constraints and generation quality. GIM-trained small models (e.g., 4B) achieve superior structural adherence (outperforming GPT-5.2) while maintaining high semantic coherence; (2) **Effective Knowledge Mining:** GIM offers a robust alternative to fragile prompting pipelines and existing information extraction methods across real-world applications, including extracting explicit static information (e.g., resume parsing), mining implicit dynamic intelligence (e.g., structured Chain-of-Thought patterns), scientific entity-relation extraction, and code infilling. We release the GIMKit system, benchmarks, and trained models[1].

1 Introduction

Large Language Models (LLMs) essentially serve as implicit containers storing world knowledge.

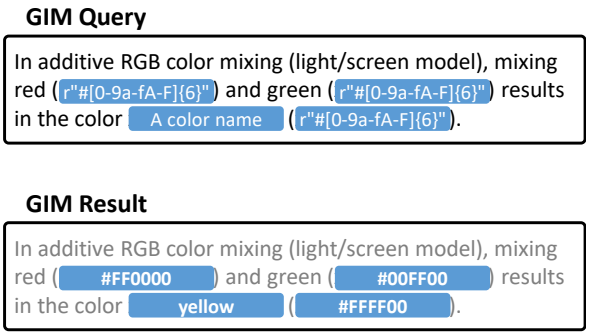

Figure 1: Input-Output example of GIM.

Previous "LM-as-KB" research (Petroni et al., 2019) has confirmed that LLMs can function as knowledge bases for storing extensive facts. However, transforming this potential into industrial-grade data mining tools is hindered by a fundamental **impedance mismatch**: LLM generation is inherently probabilistic and unstructured (Context-driven) (Liang et al., 2024b), whereas knowledge discovery pipelines strictly require deterministic, structured data formats (Schema-driven). General information extraction is essentially the intersection of *Context* (semantic background, e.g., "mixing red and green" in Figure 1) and *Schema* (structural constraints, e.g., "Hex Color Code"). The current challenge lies in enforcing strict schema compliance without compromising the model's semantic generation capabilities.

Existing alignment methods often fall short. Naive prompting, which describes the schema via natural language, acts at best as unreliable "fuzzy matching," prone to hallucinations or formatting errors. Existing constrained decoding methods enforce syntactic correctness but suffer from **Semantic Disconnection**: by treating structure merely as an output filter, these methods blind the model's internal attention (Zheng et al., 2025) to the schema, severing the link between semantic intent (the "brain") and syntactic execution (the "mouth").

[1]The GIMKit system, evaluation benchmarks, and model training scripts are available at https://anonymous.4open.science/r/GIMKit, https://anonymous.4open.science/r/GIMBench, and https://anonymous.4open.science/r/GIM-research/.

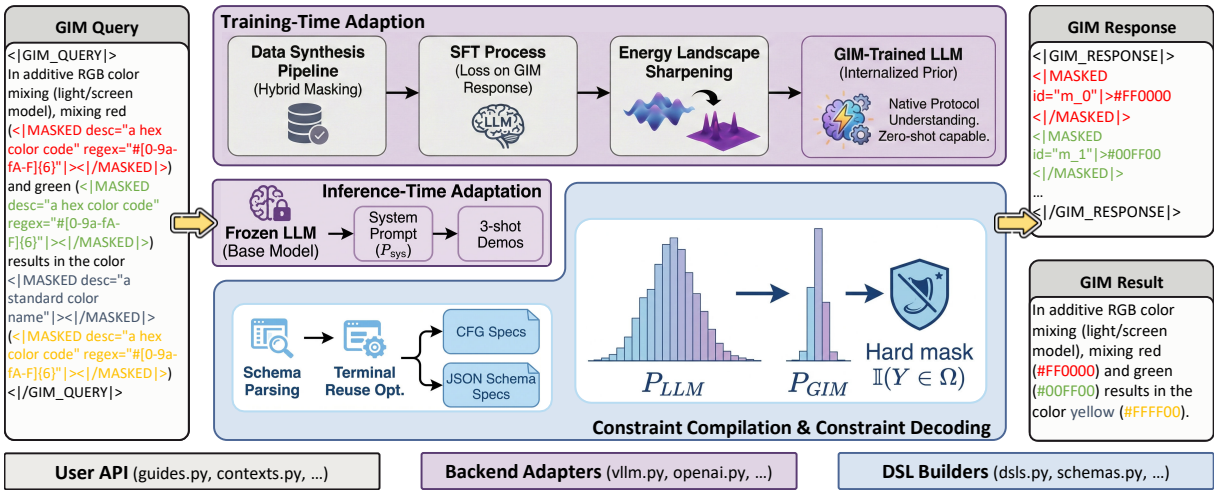

Figure 2: Guided Infilling Modeling. See Appendix A for a concrete example of how GIM elegantly resolves the color mixing problem.

To bridge this gap, we introduce **Guided Infilling Modeling (GIM)**, a novel paradigm for structured knowledge mining from LLMs. Unlike the traditional "generate-then-filter" mode, GIM reframes information extraction as a guided and constrained infilling task. We employ a **unified neuro-symbolic representation** (an XML-like GIM Query) to compile structural constraints (e.g., RegEx, types, Context-Free Grammars) directly into the semantic context (as shown in Figure 2). This approach makes the Schema an organic component of the prompt, allowing the model's self-attention to simultaneously "see" both semantic intention and structural requirements. Under this paradigm, the model is not passively "restricted" but actively "guided," performing precise **infilling** of target slots based on global structural visibility.

A common concern is whether imposing strict syntactic structure harms LLMs' pre-trained semantic capabilities. We argue that structure should be viewed as a **cognitive scaffolding**. Recent studies support this view: HtmlRAG (Tan et al., 2025) demonstrates that retaining HTML tags helps maintain semantic hierarchy and reduce hallucinations; research in the Structured Generation community (Kurt, 2024b,a) also finds that forcing structured outputs can enhance performance on general reasoning tasks. In GIM, the fusion of Context and Schema effectively "sharpens" the energy landscape of the probability distribution (as shown in the top center of Figure 2), reducing prediction entropy. This enables the stable extraction of static facts and the mining of implicit dynamic intelligence (e.g., structured Chain-of-Thought patterns).

The main contributions of this paper are as follows:

- **Paradigm Innovation:** We propose GIM, a novel paradigm that unifies structural constraints with semantic context, effectively resolving the impedance mismatch in knowledge mining from LLMs.
- **System Implementation:** We design a complete solution ranging from Inference-Time Adaptation to Training-Time Adaptation, and release the `GIMKit` system, accompanying benchmarks, and trained model weights.
- **Experimental Verification:** Extensive experiments demonstrate that GIM achieves structural adherence surpassing commercial large models while maintaining high semantic coherence. Its application value is further validated in knowledge mining scenarios such as Curriculum Vitae (CV) parsing and scientific entity-relation extraction, and even beyond, such as Chain-of-Thought (CoT) trajectory mining and code infilling.

2 Related Work

Knowledge Mining Paradigms: From Patterns to Intelligence The paradigm of Knowledge Mining has evolved from statistical pattern recognition to deep semantic understanding. Early Information Extraction (IE) relied heavily on statistical principles and expert rules, such as Apriori for frequent pattern mining (Hegland, 2005) and regex-based dictionary matching (Appelt, 1999). With the advent of deep learning, BERT-based (Devlin et al.,

2019) pipeline and joint extraction models (e.g., DeepKE (Zhang et al., 2022), CasRel (Wei et al., 2020)) significantly improved performance. Prior system-centric approaches like DeepDive (Shin et al., 2015) treated Knowledge Base Construction (KBC) as statistical inference over large-scale features, achieving high precision but requiring complex feature engineering. Recently, "LM-as-KB" studies (Petroni et al., 2019) demonstrated that pretrained models can recall facts without fine-tuning, while UIE (Lu et al., 2022) attempted to unify IE tasks via a Seq2Seq framework. However, these methods remain fragile for industrial-grade mining, often depending on unstable prompt engineering. Current mining processes are typically unstructured and limited to explicit fact triples (AlKhamissi et al., 2022). GIM bridges this gap by introducing a structured infilling paradigm with explicit constraints, moving beyond simple "probing" for static entities to guided extraction of complex information, including dynamic CoT reasoning trajectories (Wei et al., 2022).

Structured Generation and Constrained Decoding Structured Generation constrains outputs to predefined schemas. The mainstream approach is Constrained Decoding, exemplified by Outlines (Willard and Louf, 2023) and XGrammar (Dong et al., 2025). These methods employ Finite State Machines (FSM) or Prefix Trees (Park et al., 2025) to filter logits during inference, enforcing JSON Schema or Context-Free Grammar (CFG) compliance. While ensuring syntactic correctness, they suffer from **Semantic Disconnection**. Although recent works attempt to internalize structure via synthetic data training (Godoy, 2025), they are often task-specific. In contrast, GIM leverages Training-Time Adaptation and explicit schema embedding, enabling the model to perceive global structural constraints *before* generation, thus aligning structure while maintaining semantic coherence.

Fill-in-the-Middle (FIM) and Text Infilling Text Infilling, or the Cloze task, serves as the technical foundation of GIM. Tracing back to BERT's Masked Language Modeling (MLM) (Devlin et al., 2019), this task was initially limited by Encoder architectures unable to generate arbitrary-length text. Subsequently, autoregressive infilling methods based on Decoder-only architectures were proposed (Zhu et al., 2019; Donahue et al., 2020), predicting middle content by concatenating prefixes and suffixes. InCoder (Fried et al., 2023) and OpenAI (Bavarian et al., 2022) extended FIM to large-scale code generation, proving that FIM training enhances structural understanding; it has since become standard in models like StarCoder (Li et al., 2023) and Code Llama (Rozière et al., 2023). From an information-theoretic perspective, traditional Left-to-Right generation is divergent (prone to high entropy), whereas FIM utilizes bidirectional constraints to make the process probabilistically convergent. GIM generalizes this infilling paradigm to general data mining.

3 Guided Infilling Modeling

3.1 Problem Formulation

GIM reformulates natural language generation as a **Constrained Probabilistic Generation** task over a formal language space. Unlike unconstrained autoregressive generation, GIM explicitly embeds the schema into the query context, unifying a dual-guidance mechanism: **semantic guidance** via natural language to shape the probabilistic energy landscape, and **structural guidance** via formal grammar to strictly define the valid support set.

Let $\mathcal{V}$ be the vocabulary and $\mathcal{V}^*$ be the Kleene closure of all possible token sequences. An LLM represents a parameterized distribution $P_\theta(Y \mid X)$ over $Y \in \mathcal{V}^*$ conditioned on context X.

Definition 1 (GIM Query)**.** *A GIM Query* $\mathbf{Q}$ *is a sequence of segments* $\mathbf{Q} = \langle s_1, s_2, \ldots, s_N \rangle$*. Each segment* s_k *is either fixed natural language context* ($s_k \in \mathcal{T}$) *or a* ***Masked Slot*** ($s_k \in \mathcal{S}$)*. Each masked slot* $m \in \mathcal{S}$ *is defined as a dual-guided tuple:*

$$m = \langle \mathcal{D}_{sem}, \mathcal{G}_{str} \rangle \tag{1}$$

where:

- $\mathcal{D}_{sem}$ *(Semantic Guidance): A natural language description (e.g., "candidate name") that becomes part of the conditional context, influencing the model's intention.*
- $\mathcal{G}_{str}$ *(Structural Guidance): A formal grammar (e.g., RegEx or CFG) defining the set of syntactically valid strings* $L(\mathcal{G}_{str}) \subset \mathcal{V}^*$*, strictly restricting the format.*

Definition 2 (The GIM Distribution)**.** *To unify this dual guidance, we construct the* ***GIM Distribution*** P_{GIM}*, a renormalized probability distribution projecting the LLM's intrinsic knowledge onto the intersection of semantics and structure.*

Let $\mathbf{C}(\mathbf{Q})$ be the flattened semantic context and $\Omega(\mathbf{Q})$ be the global structural constraint. Then:

$$P_{GIM}(Y \mid \mathbf{Q}) = \frac{P_\theta(Y \mid \mathbf{C}(\mathbf{Q})) \cdot \mathbb{I}(Y \in L(\Omega(\mathbf{Q})))}{Z(\mathbf{Q})} \tag{2}$$

where $\mathbb{I}(\cdot)$ is the indicator function for hard structural filtering, and $Z(\mathbf{Q})$ is the partition function ensuring normalization over the valid support set.

While $Z(\mathbf{Q})$ can be approximated via Monte Carlo Sampling, inference primarily seeks the optimal sequence Y^* rather than exact probabilities. Leveraging the scale-invariance of $Z(\mathbf{Q})$ with respect to candidate Y, we bypass expensive normalization and cast generation as a **Constrained Maximum A Posteriori (MAP)** optimization:

$$\begin{aligned} Y^* &= \arg\max_{Y \in \mathcal{V}^*} P_{\text{GIM}}(Y \mid \mathbf{Q}) \\ &= \arg\max_{Y \in \mathcal{V}^*} \sum_{t=1}^{|Y|} \log P_\theta(y_t \mid y_{<t}, \mathbf{C}(\mathbf{Q})) \\ &\quad \text{s.t.} \quad \forall m_k \in \mathcal{S}, \quad \text{Extract}(Y, m_k) \in L(\mathcal{G}_{str}^{(k)}) \end{aligned} \tag{3}$$

This formulation can also be interpreted through a database query lens, as discussed in Appendix B.

3.2 Constraint Compilation

To efficiently solve the Constrained MAP problem (Eq. 3), we require a mechanism to translate logical constraints $\Omega(\mathbf{Q})$ into executable token-level masks. We introduce the Constraint Compiler Φ, which translates high-level GIM protocols into CFG or JSON Schema.

Global Grammar Construction. The compiler first builds a global Start Rule based on the query topology. Unlike traditional local constraints, the GIM compiler treats static text and Masked Slots as a unified whole, forcing the model to generate protocol-defined structural tags (e.g., `<|MASKED...>`) correctly. This "global visibility" ensures generation strictly adheres to the predefined schema structure.

Optimization via Terminal Reuse. In large-scale extraction, repetitive structural constraints (e.g., multiple date fields sharing the same RegEx) cause state space redundancy. We implement a Terminal Reuse mechanism where the compiler maintains a symbol table to map semantically equivalent regexes to shared Terminal Rules. As shown in Figure 3, although m_0, m_1, m_3 correspond to different slots, they share the same terminal T_0.

For closed-source models lacking logit intervention support, the compiler provides a fallback by transpiling protocols into JSON Schema (details in Appendix C).

```
%llguidance {}
start:
    "<|GIM_RESPONSE|>"
    REGEX "<|MASKED id=\"m_0\"|>" m_0
    REGEX "<|MASKED id=\"m_1\"|>" m_1
    REGEX "<|MASKED id=\"m_2\"|>" m_2
    REGEX "<|MASKED id=\"m_3\"|>" m_3
    REGEX "<|/GIM_RESPONSE|>"
REGEX: /\s*/
m_0[capture, suffix="<|/MASKED|>"]: T_0
m_1[capture, suffix="<|/MASKED|>"]: T_0
m_2[capture, suffix="<|/MASKED|>"]: T_1
m_3[capture, suffix="<|/MASKED|>"]: T_0
T_0: /#[0-9a-fA-F]{6}/
T_1: /(?s:.*)/
```

Figure 3: Example CFG for color mixing problem in Figure 1. Terminal T_0 is shared by three slots, demonstrating Terminal Reuse.

3.3 Model Adaptation Strategy

GIM is a model-agnostic paradigm. Any system capable of performing structured infilling tasks under contextual constraints can be viewed as an implementation. We design adaptation strategies for both training and inference phases.

3.3.1 Training-Time: Shaping the Prior

To acquire native understanding of GIM protocols and internalize syntactic rules, we explore Supervised Fine-Tuning. By tuning on constructed GIM Query/Response pairs, the model learns the "infilling" paradigm.

Training Objective. We optimize using the standard Causal Language Modeling objective, restricting loss calculation to the GIM Response sequence:

$$\mathcal{L}(\theta) = -\sum_{t \in \text{GIM Response}} \log P_\theta(y_t \mid \mathbf{x}, y_{<t}) \tag{4}$$

Training Data Synthesis & Recipe. Leveraging the self-supervision-friendly nature of GIM, we synthesize training data via hybrid masking on high-quality corpora: (1) **Poisson Random Masking** forces the model to learn general context recovery; (2) **Semantic/Reasoning Masking**

specifically masks CoT and conclusions, guiding the model to perform structured reasoning.

We constructed the `GIM-SFT` dataset containing 3,157,829 bilingual samples (see Table 1). Through extensive ablation on data recipes, we adopted a strategy retaining all high-quality subsets (e.g., HK-O1aw, LIMA) while downsampling low-quality data, resulting in 50k samples. We trained **`GIM-1.7B`** and **`GIM-4B`** based on the Qwen3 architecture (Yang et al., 2025). Ablation studies validating the masking strategy and the importance of high-quality subsets are provided in Appendix G.6.

Synergy with Constrained Decoding. Notably, while SFT significantly aligns the model's prior, we still employ compiled CFGs during inference to ensure zero syntax errors. Unlike Inference-Time Adaptation (described next), GIM-trained models respond directly to GIM Queries without System Prompt or Few-shot Demonstrations. This combination of "Internalized Prior + Forced Posterior" guarantees 100% structural adherence in theory.

Data Source	Count
`HKAIR-Lab/HK-O1aw-SFT-16K` (Lab, 2024)	14,363
`Ki-Seki/UHGEvalDataset` (Liang et al., 2024a)	5,141
`thesven/gsm8k-reasoning` (Svendsen, 2024)	1,254
`Qwen/ProcessBench` (Zheng et al., 2024)	1,179
`GAIR/lima` (Zhou et al., 2023)	1,030
`GAIR/o1-journey` (Qin et al., 2024)	327
`abisee/cnn_dailymail` (See et al., 2017)	287,113
`Magpie-Reasoning` (Xu et al., 2024)	150,000
`kaist-ai/CoT-Collection` (Kim et al., 2023)	1,837,928
`AI-MO/NuminaMath-CoT` (LI et al., 2024)	859,494

Table 1: Statistics of Training Data. Datasets are grouped by their original quality, high, medium, and low from top to bottom. Magpie-Reasoning is short for Magpie-Align/Magpie-Reasoning-150K.

3.3.2 Inference-Time: Constraining the Posterior

For scenarios where parameter updates are infeasible, we implement GIM by intervening in the decoding process using compiler-generated constraints. To compensate for the lack of GIM protocol knowledge, we construct a System Prompt of protocol specification ($\mathcal{P}_{sys}$) and Few-shot Demonstrations ($\mathcal{D}_{demo}$) (see Appendix D). The generation distribution approximates:

$$P(Y \mid \mathbf{Q}) \approx P_{\text{LLM}}(Y \mid \mathcal{P}_{sys}, \mathcal{D}_{demo}, \mathbf{Q}) \cdot \mathbb{I}(Y \in \Omega(\mathbf{Q})) \tag{5}$$

where $\mathcal{D}_{demo}$ includes three standard pairs of (GIM Query, GIM Response) to activate protocol adherence via In-Context Learning. We implement two paths based on model access permissions:

Open-Weights Models via CFG Decoding. For open-source models with logit access (e.g., Llama, Qwen), GIMKit compiles the schema into a CFG and attaches a Logits Processor. Although untrained, the base model's instruction-following capability, combined with mandatory syntax masking, achieves high-precision structured infilling.

Closed-Source Models via JSON Schema. For API-only models (e.g., GPT-4, Gemini), we design a fallback using JSON Schema. The compiled schema forces JSON output via the API. While this sacrifices deep understanding of semantic guidance ($\mathcal{D}_{sem}$), its ecosystem compatibility allows "plug-and-play" application of GIM to mainstream commercial models.

3.4 The GIMKit System Architecture

To bridge the gap between theoretical paradigm and industrial deployment, we release the industrial-grade GIMKit system. As a Python reference implementation, GIMKit adopts a modular architecture containing three core components (as shown in Figure 2):

- **User API:** Provides a Pythonic programming interface (`gimkit.guide`). Users can define Masked Tags declaratively using Type Hints and RegEx, which the system automatically serializes into standard GIM Queries.
- **DSL Builders:** The core engine responsible for parsing high-level GIM Queries and compiling them into execution-ready constraint specifications. It features built-in terminal generation algorithms and state machine optimization logic.
- **Backend Adapters:** This component abstracts the differences between underlying inference engines. It directly injects compiled CFGs for local engines (e.g., vLLM) or falls back to Prompt and JSON Schema simulation for remote APIs.

4 Experiments

4.1 Experimental Setup

Since guided infilling explicitly intervenes in the natural language generation order via global constraints, it diverges from traditional unconstrained

autoregressive modeling. Therefore, we assess the GIM paradigm across two dimensions: **Infilling Quality** (e.g., Normalized Perplexity) and **Protocol Adherence** (e.g., RegEx Match Rate), to verify whether the paradigm maintains intrinsic semantic coherence while achieving structural control.

4.1.1 Normalized Perplexity on `GIM-SFT` Test Set

To assess the linguistic modeling quality of guided infilling, we employ a fill-in-the-blank evaluation on a test subset of `GIM-SFT`. Specifically, model predictions are filled back into the query context to constitute a mask-free GIM Result $\mathcal{R}$. We compute perplexity-based metrics on substrings of $\mathcal{R}$ using a reference language model Q_{ref} (e.g., `google/gemma-3-270m`).

Since raw perplexity is unbounded and highly vocabulary-dependent, making direct comparisons across tokenizers difficult (Roh et al., 2020), we follow existing work and normalize perplexity against a random baseline model ($\text{PPL}_{\text{rand}} = |\mathcal{V}|$)[2].

For a token sequence with cross-entropy loss $\mathcal{L}$ under Q_{ref}, we define **Normalized Perplexity (NP)** as:

$$\text{NP} = \left(\frac{\text{PPL}_{\text{ref}}}{\text{PPL}_{\text{rand}}}\right)^{\alpha} = \left(\frac{\exp(\mathcal{L})}{|\mathcal{V}|}\right)^{\alpha} \quad (6)$$

where $\alpha = 0.2$ is a scaling factor to map values to a stable range.

Furthermore, as perplexity is biased by sequence length (Li et al., 2024), we use **Windowed Normalized Perplexity (WNP)** to strictly evaluate local context consistency. For each infilled span, WNP is calculated independently within a fixed radius k around the span boundaries.

4.1.2 Constraint Match Rate on `GIMBench-regex-match` Dataset

Beyond linguistic quality, a core capability of GIM is **protocol adherence**. Masked slots in GIM often carry explicit RegEx constraints. We designed a metric suite to evaluate controllability and reliability under these constraints.

We constructed `GIMBench-regex-match` (961 samples) by prompting seven advanced LLMs to generate articles with random masks and corresponding RegEx patterns based on seed keywords. Using GIMKit, we measure performance across two dimensions:

Prediction Rate measures the model's willingness to generate valid predictions for masked slots. It is calculated as the ratio of non-empty filled tags to the total number of masked tags in a query, reflecting whether the model actively participates in the constrained task rather than evading it.

RegEx Match Rate measures precise adherence to explicit constraints. For all tags with RegEx constraints where the model generated content, we perform strict full matching (`re.fullmatch`). The final rate is the ratio of successfully matched tags to the total number of valid constrained tags.

4.1.3 Experimental Models

We evaluate representative state-of-the-art models for Inference-Time Adaptation and our proprietary `GIM-1.7B` and `GIM-4B` for Training-Time Adaptation. See Table 3 in Appendix E for details.

4.2 Experimental Results Analysis

	Avg. $\text{WNP}_{k=16}$ ↓	Pred. Rate ↑	Match Rate ↑
claude-haiku-...	36.75%$_{(1)}$	100.00%$_{(1)}$	49.41%$_{(15)}$
gemini-3-flas...	36.82%$_{(2)}$	100.00%$_{(1)}$	96.27%$_{(1)}$
gpt-5.2	38.42%$_{(11)}$	96.06%$_{(11)}$	83.21%$_{(3)}$
grok-4-fast	37.47%$_{(7)}$	99.75%$_{(3)}$	79.15%$_{(4)}$
deepseek-chat...	38.72%$_{(13)}$	98.41%$_{(4)}$	70.69%$_{(9)}$
minimax-m2.1	38.62%$_{(12)}$	94.36%$_{(13)}$	60.94%$_{(12)}$
kimi-k2-think...	39.17%$_{(14)}$	97.51%$_{(5)}$	47.39%$_{(16)}$
mimo-v2-flash	37.01%$_{(4)}$	96.48%$_{(9)}$	65.50%$_{(10)}$
gemma-3-12b-i...	37.24%$_{(6)}$	96.76%$_{(7)}$	58.33%$_{(13)}$
gemma-3-27b-i...	37.85%$_{(9)}$	95.86%$_{(12)}$	62.15%$_{(11)}$
ministral-14b...	39.57%$_{(15)}$	96.60%$_{(8)}$	77.02%$_{(6)}$
qwen3-30b-a3b...	37.17%$_{(5)}$	97.35%$_{(6)}$	57.76%$_{(14)}$
Qwen3-1.7B	40.39%$_{(16)}$	91.24%$_{(15)}$	73.15%$_{(8)}$
Qwen3-4B-Inst...	37.95%$_{(10)}$	86.11%$_{(16)}$	74.19%$_{(7)}$
GIM-1.7B	37.47%$_{(8)}$	93.87%$_{(14)}$	77.04%$_{(5)}$
GIM-4B	36.84%$_{(3)}$	96.35%$_{(10)}$	83.78%$_{(2)}$

Rank: 1 4 8 12 16

Figure 4: Performance Heatmap across Three Metrics.

Figure 4 illustrates the performance comparison between GIM and baselines. The data strongly validates the feasibility of the GIM paradigm.

Model-Agnostic Robustness Experiments covering a broad spectrum from closed-source SOTA (e.g., `grok-4-fast`, `gemini-3-flash-preview`) to open-weights models (e.g., `mimo-v2-flash`) demonstrate that GIM serves as a universal protocol. Whether via Inference-Time or Training-Time

[2]For a random model with a vocabulary size $|\mathcal{V}|$, the probability of predicting any specific token x_i is uniform: $P(x_i) = \frac{1}{|\mathcal{V}|}$. Therefore, $\text{PPL}_{\text{rand}} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\ln\frac{1}{|\mathcal{V}|}\right) = \exp\left(-\frac{1}{N}\cdot N\cdot(-\ln|\mathcal{V}|)\right) = \exp(\ln|\mathcal{V}|) = |\mathcal{V}|$.

Adaptation, GIM effectively drives structured infilling across diverse architectures.

Synergy in Training-Time Adaptation Comparing the `Qwen3` series with their GIM-trained counterparts reveals significant gains. For instance, `GIM-4B` achieves superior WNP (Rank 10 → Rank 3) alongside a massive boost in Match Rate compared to `Qwen3-4B-Inst`. This indicates that GIM training does not merely "prune" the generation space but optimizes the synergy between structural constraints and semantic fluency by reducing prediction entropy through dual guidance.

Efficiency: Superior Performance per Parameter Notably, GIM empowers small models with industrial-grade structural adherence. The 4B-parameter `GIM-4B` achieves an 83.78% Match Rate (Rank 2), significantly outperforming same-size models and even surpassing `gpt-5.2` (83.21%) and `ministral-14b` (77.02%). This strongly demonstrates that the GIM paradigm effectively concentrates probability density onto the schema-compliant manifold, enabling edge-side small models to perform high-precision structured extraction.

Limitations of Inference-Only Adaptation While models like claude-haiku excel in WNP (Rank 1) due to strong linguistic foundations, they suffer from low Match Rates (49.41%, Rank 15). This "High Semantics, Low Adherence" phenomenon highlights the insufficiency of general instruction following for strict schema constraints and underscores the necessity of GIM training for high-precision mining tasks.

Parameter Scaling Within GIM models, `GIM-4B` consistently outperforms `GIM-1.7B` across all metrics. This suggests the GIM paradigm adheres to standard scaling laws, with joint modeling capabilities for complex structure and semantics improving as parameter size increases.

4.3 Application Study: CV parsing

We now demonstrate GIM's practical value in resolving the impedance mismatch through static information extraction under high noise: CV parsing.

4.3.1 Experimental Setup

Data Construction We constructed the `GIMBench-cv-parse` dataset derived from the recruitment scenario of a large internet company (Company A[3]), where one of the authors is an intern. The dataset comprises 43 authorized resume PDFs. Text was extracted using MinerU (Niu et al., 2025), followed by multi-round human verification to annotate 472 fine-grained fields as ground truth.

[3]Company A is anonymized for review purposes.

Model	w/o GIM	w/ GIM	Δ
claude-haiku-4.5	0.00%	87.50%	**+87.50%**
gemini-3-flash-preview	78.81%	77.97%	–0.84%
gpt-5.2	74.15%	71.19%	–2.96%
grok-4-fast	68.86%	70.76%	**+1.90%**
deepseek-chat-v3-0324	66.31%	72.25%	**+5.94%**
minimax-m2.1	72.25%	69.07%	–3.18%
kimi-k2-thinking	75.21%	74.65%	–0.56%
mimo-v2-flash	71.82%	73.52%	**+1.70%**
gemma-3-12b-it	58.22%	71.37%	**+13.15%**
gemma-3-27b-it	67.37%	70.34%	**+2.97%**
ministral-14b-2512	65.89%	62.17%	–3.72%
qwen3-30b-a3b-instruct	68.22%	75.64%	**+7.42%**
Qwen3-1.7B	63.77%	59.96%	–3.81%
Qwen3-4B-Instruct	12.50%	68.83%	**+56.33%**
PP-UIE-7B	46.19%	N/A	–
gliner2-large-v1	44.49%	N/A	–
GIM-1.7B	N/A	62.29%	N/A
GIM-4B	N/A	66.31%	N/A

Table 2: Comparison of Extraction Accuracy on Real-World CV Parsing.

Baseline We utilize the model set listed in Table 3. We define the industry-standard structural extraction approach as the baseline: **Standard Prompting + JSON Schema**. Here, inputs are loosely coupled: a prompt describes the extraction intent (Context), while a standard JSON Schema serves merely as a hard formatting constraint. We also employ traditional information-extraction SOTA models, including PP-UIE-7B[4] and gliner2-large-v1 (Zaratiana et al., 2025).

GIM Implementation In contrast, the **GIM** paradigm refactors this process via Inference-Time Adaptation. GIM compiles the schema definition directly into the prompt context, forcing guided Infilling (making structure "visible" within the context). See Appendix F for template differences.

4.3.2 Results and Analysis

As shown in Table 2, GIM demonstrates significant advantages. Excluding GIM-trained models, 8 out of 14 general-purpose models achieved performance gains upon adopting GIM. The experiments reveal three key findings:

[4]https://aistudio.baidu.com/modelsdetail/25875/intro

Bridging the Disconnection Baselines often suffer from the separation of semantic instructions and format constraints. For instance, when extracting "highest degree," while the prompt specifies "highest," the JSON Schema only constrains the data type. This physical separation causes information loss, leading to syntactically correct but semantically erroneous outputs (e.g., listing a non-highest degree). By embedding constraints at the generation site, GIM forces the attention mechanism to attend to both semantic intent and structural boundaries simultaneously, enhancing accuracy.

Mitigating Formatting Collapse `claude-haiku` and `Qwen3-4B` failed catastrophically under the Baseline (0.00% and 12.50%) primarily due to loss of control over JSON syntax, resulting in unparsable outputs.

Gap Analysis for GIM-Trained Models We observed that `GIM-Trained` models did not achieve SOTA on this task. We initially suspected the cause to be a context length distribution mismatch: CV content averages 2152.05 tokens, whereas training queries average only 251.48 tokens. However, training on longer sequences did not resolve the issue—it instead led to a slight *decrease* in Match Rate, ruling out length as the primary factor. Further ablation studies revealed that the root cause lies in **training data quality**: high-quality *Semantic/Reasoning Masking* samples—where masked slots require non-trivial inference—contribute disproportionately to model performance, and removing them causes a significant drop in Match Rate. The current training corpus lacks sufficient coverage of such samples. We are actively working to address this data gap. Full ablation details are provided in Appendix G.

Engineering Impact Appendix F compares prompts for both paradigms. GIM provides a **Unified Neuro-Symbolic Interface**, eliminating the complexity of maintaining separate JSON Schemas and making extraction logic more intuitive and maintainable.

4.4 Further Experiments

Beyond the main experiments, we conducted a series of additional studies whose full reports are provided in the appendix. GIM consistently demonstrates strong effectiveness across three further application domains:

- **Code Infilling** (Appendix G.1): On HumanEval-Infilling, GIM-trained models substantially outperform their base counterparts (`GIM-1.7B`: $+17.5$ pp; `GIM-4B`: $+20.5$ pp pass@1), showing that the infilling paradigm generalizes to syntactically sensitive code structures.

- **Reasoning Trajectory Mining** (Appendix G.2): GIM mines structured step-aligned CoT trajectories without degrading reasoning accuracy (e.g., `gemma-3-12b-it` achieves 80.34% on QASC under GIM, comparable to free CoT at 79.59%).

- **Relation Extraction on SciERC** (Luan et al., 2018) (Appendix G.3): `GIM-4B` achieves a **0% format error rate**—the only model to do so—and attains the highest Strict F1 (5.42%) on this complex graph-structured schema benchmark.

We also report an overhead and cost analysis in Appendix G.4 and two ablation studies: an XML representation ablation (Appendix G.5) and a training data ablation (Appendix G.6).

All experiments in this paper are conducted in the same computational environment as detailed in Appendix H to ensure reproducibility.

5 Conclusion

This paper introduces Guided Infilling Modeling (GIM), a unified paradigm integrating structural constraints with semantic context to resolve the impedance mismatch between probabilistic generation and deterministic knowledge mining. By compiling schemas directly into the generation process, GIM enables structured, constraint-guided extraction from LLMs. Extensive experiments demonstrate that this "structural guidance" does not compromise semantic coherence; instead, by sharpening the probability distribution, it enables a 4B-parameter GIM model to outperform commercial SOTA models like GPT-5.2 in structural adherence. GIM significantly enhances robustness in static CV parsing and demonstrates the potential to extract high-quality reasoning trajectories in Chain-of-Thought mining. We release the GIMKit system to provide the community with a maintainable infrastructure for neuro-symbolic reasoning.

Limitations

Despite its potential in structured mining, GIM still faces challenges regarding the "halting problem" and the trade-off with Minimal Description Length in open-ended generation. Future work will explore integrating Reinforcement Learning to further align the model's preference for conciseness with schema constraints. In this era of rapidly evolving intelligence, GIM offers a novel technical pathway for the community to stably mine explicit knowledge from vast probabilistic distributions.

The aforementioned scarcity of high-quality training samples remains an open challenge; we plan to address this through large-scale synthetic data generation in future work.

Ethical considerations

All public datasets used in this study comply with their original licenses and intended-use policies. No modifications were made that would violate the terms of distribution. A portion of the data consists of non-public documents that were accessed under proper authorization within the organization. These materials were processed to remove any personally identifiable information and are used solely for internal research. In accordance with organizational policy and privacy requirements, these data will not be released or distributed. All annotations used in this work were created by the authors themselves. No external annotators were involved, and no human subjects were recruited for data collection. Therefore, no additional consent procedures or compensation protocols were required. All model artifacts and code used in this work adhere to their respective licenses and are permitted for research and commercial use where applicable.

References

Badr AlKhamissi, Millicent Li, Asli Celikyilmaz, Mona Diab, and Marjan Ghazvininejad. 2022. A review on language models as knowledge bases. *Preprint*, arXiv:2204.06031.

Anthropic. 2025. Claude haiku 4.5. https://www.anthropic.com/claude/haiku. Accessed: 2026-02-04.

Douglas E. Appelt. 1999. Introduction to information extraction. *AI Communications*, 12(3):161–172.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. 2022. Efficient training of language models to fill in the middle. *Preprint*, arXiv:2207.14255.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, and 1 others. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Chris Donahue, Mina Lee, and Percy Liang. 2020. Enabling language models to fill in the blanks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2492–2501, Online. Association for Computational Linguistics.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. 2025. Xgrammar: Flexible and efficient structured generation engine for large language models. *Preprint*, arXiv:2411.15100.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations*.

Henrique Godoy. 2025. Extract-0: A specialized language model for document information extraction. *Preprint*, arXiv:2509.22906.

Markus Hegland. 2005. The apriori algorithm–a tutorial. *Mathematics and computation in imaging science and information processing*.

Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. 2020. Qasc: A dataset for question answering via sentence composition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8082–8090.

Seungone Kim, Se June Joo, Doyoung Kim, Joel Jang, Seonghyeon Ye, Jamin Shin, and Minjoon Seo. 2023. The cot collection: Improving zero-shot and few-shot learning of language models via chain-of-thought fine-tuning. *arXiv preprint arXiv:2305.14045*.

Will Kurt. 2024a. Say what you mean: A response to 'let me speak freely'. https://blog.dottxt.ai/say-what-you-mean.html. Accessed September 15, 2025.

Will Kurt. 2024b. Structured generation improves llm performance: Gsm8k benchmark. https://blog.dottxt.ai/performance-gsm8k.html. Accessed September 15, 2025.

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

HKAIR Lab. 2024. Hk-o1aw models: Leveraging o1 slow thinking in the development of hong kong legal large language models. https://github.com/HKAIR-Lab/HK-O1aw.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. 2024. Numinamath. [https://huggingface.co/AI-MO/NuminaMath-CoT](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf).

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, and 48 others. 2023. Starcoder: may the source be with you! *Preprint*, arXiv:2305.06161.

Zhenyu Li, Yike Zhang, Tengyu Pan, Yutao Sun, Zhichao Duan, Junjie Fang, Rong Han, Zixuan Wang, and Jianyong Wang. 2024. Focusllm: Precise understanding of long context by dynamic condensing. *Preprint*, arXiv:2408.11745.

Xun Liang, Shichao Song, Simin Niu, Zhiyu Li, Feiyu Xiong, Bo Tang, Yezhaohui Wang, Dawei He, Cheng Peng, Zhonghao Wang, and Haiying Deng. 2024a. UHGEval: Benchmarking the hallucination of Chinese large language models via unconstrained generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5266–5293, Bangkok, Thailand. Association for Computational Linguistics.

Xun Liang, Shichao Song, Zifan Zheng, Hanyu Wang, Qingchen Yu, Xunkai Li, Rong-Hua Li, Yi Wang, Zhonghao Wang, Feiyu Xiong, and 1 others. 2024b. Internal consistency and self-feedback in large language models: A survey. *arXiv preprint arXiv:2407.14507*.

Alexander H. Liu, Kartik Khandelwal, Sandeep Subramanian, and 1 others. 2026. Ministral 3. *Preprint*, arXiv:2601.08584.

Yaojie Lu, Qing Liu, Dai Dai, Xinyan Xiao, Hongyu Lin, Xianpei Han, Le Sun, and Hua Wu. 2022. Unified structure generation for universal information extraction. *arXiv preprint arXiv:2203.12277*.

Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. 2018. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. *Preprint*, arXiv:1808.09602.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. 2024. Improve mathematical reasoning in language models by automated process supervision. *Preprint*, arXiv:2406.06592.

MiniMax. 2025. Minimax m2.1: Significantly enhanced multi-language programming, built for real-world complex tasks. https://www.minimax.io/news/minimax-m21. Accessed: 2026-02-04.

Junbo Niu, Zheng Liu, Zhuangcheng Gu, Bin Wang, Linke Ouyang, Zhiyuan Zhao, Tao Chu, Tianyao He, Fan Wu, Qintong Zhang, Zhenjiang Jin, Guang Liang, Rui Zhang, Wenzheng Zhang, Yuan Qu, Zhifei Ren, Yuefeng Sun, Yuanhong Zheng, Dongsheng Ma, and 42 others. 2025. Mineru2.5: A decoupled vision-language model for efficient high-resolution document parsing. *Preprint*, arXiv:2509.22186.

OpenAI. 2026. Introducing gpt-5.2. https://openai.com/index/introducing-gpt-5-2. Accessed: 2026-02-04.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. 2022. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Conference on health, inference, and learning*, pages 248–260. PMLR.

Sihyeong Park, Sungryeol Jeon, Chaelyn Lee, Seokhun Jeon, Byung-Soo Kim, and Jemin Lee. 2025. A survey on inference engines for large language models: Perspectives on optimization and efficiency. *Preprint*, arXiv:2505.01658.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Yiwei Qin, Xuefeng Li, Haoyang Zou, Yixiu Liu, Shijie Xia, Zhen Huang, Yixin Ye, Weizhe Yuan, Zhengzhong Liu, Yuanzhi Li, and Pengfei Liu. 2024. O1 replication journey: A strategic progress report – part 1. https://github.com/GAIR-NLP/O1-Journey.

Jihyeon Roh, Sang-Hoon Oh, and Soo-Young Lee. 2020. Unigram-normalized perplexity as a language model performance measure with different vocabulary sizes. *Preprint*, arXiv:2011.13220.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna

Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2023. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. 2015. Incremental knowledge base construction using deepdive. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, volume 8, page 1310.

Michael Svendsen. 2024. thesven/gsm8k-reasoning. https://huggingface.co/datasets/thesven/gsm8k-reasoning. Accessed: 2026-02-04.

Jiejun Tan, Zhicheng Dou, Wen Wang, Mang Wang, Weipeng Chen, and Ji-Rong Wen. 2025. Htmlrag: Html is better than plain text for modeling retrieved knowledge in rag systems. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 1733–1746, New York, NY, USA. Association for Computing Machinery.

Core Team, Bangjun Xiao, Bingquan Xia, Bo Yang, and 1 others. 2026a. Mimo-v2-flash technical report. *Preprint*, arXiv:2601.02780.

Gemma Team, Aishwarya Kamath, Johan Ferret, and 1 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Kimi Team, Yifan Bai, Yiping Bao, Y. Charles, and 1 others. 2026b. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zhepei Wei, Jianlin Su, Yue Wang, Yuan Tian, and Yi Chang. 2020. A novel cascade binary tagging framework for relational triple extraction. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 1476–1488.

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

xAI. 2025. Grok 4 fast. https://x.ai/news/grok-4-fast. Accessed: 2026-02-04.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. 2024. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing.

An Yang, Anfeng Li, Baosong Yang, and 1 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Urchade Zaratiana, Gil Pasternak, Oliver Boyd, George Hurn-Maloney, and Ash Lewis. 2025. GLiNER2: Schema-driven multi-task learning for structured information extraction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 130–140, Suzhou, China. Association for Computational Linguistics.

Ningyu Zhang, Xin Xu, Liankuan Tao, Haiyang Yu, Hongbin Ye, Shuofei Qiao, Xin Xie, Xiang Chen, Zhoubo Li, Lei Li, and 1 others. 2022. Deepke: A deep learning based knowledge extraction toolkit for knowledge base population. *arXiv preprint arXiv:2201.03335*.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2024. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*.

Zifan Zheng, Yezhaohui Wang, Yuxin Huang, Shichao Song, Mingchuan Yang, Bo Tang, Feiyu Xiong, and Zhiyu Li. 2025. Attention heads of large language models. *Patterns*, 6(2).

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. Lima: Less is more for alignment. *Preprint*, arXiv:2305.11206.

Wanrong Zhu, Zhiting Hu, and Eric Xing. 2019. Text infilling. *Preprint*, arXiv:1901.00158.

Appendix Contents

A Color Mixing Example **13**

B SQL Perspective on GIM **13**

C Constraint Compilation Details **13**
C.1 CFG Compilation Logic 13
C.2 JSON Schema Transpilation 14

D System Prompt and Demonstrations **14**

E Experimental Models **15**

F Template Comparison for CV Parsing **15**

G Further Experiments **15**
G.1 Code Infilling: HumanEval-Infilling 15
G.2 Structured Reasoning Trajectories Mining 15
G.3 Relation Extraction on SciERC 16
G.4 Decoding Throughput Analysis 17
G.5 XML Representation Ablation 17
G.6 Training Data Ablation Studies 18

H Computational Resources **18**

A Color Mixing Example

For the color mixing problem, the implementation using GIMKit is shown in Listing 1, and the corresponding output is presented in Listing 2.

```python
from vllm import LLM, SamplingParams

from gimkit import from_vllm_offline
from gimkit import guide as g


# 1. Use A Model
llm = LLM(model="<Release after acceptance>", max_model_len=8192)
model = from_vllm_offline(llm)

# 2. Define A Query With Guide
hex_color = g(desc="a hex color code", regex=r"#[0-9a-fA-F]{6}")
color_name = g(desc="a standard color name")

query = (
    f"In additive RGB color mixing (light/screen model), "
    f"mixing red ({hex_color}) and green ({hex_color}) "
    f"results in the color {color_name} ({hex_color})."
)
print(f"GIM Query: {query}\n\n")

# 3. Get The Result
result = model(query, output_type="cfg", sampling_params=SamplingParams(max_tokens=500, temperature=0.0))
print(f"GIM Result: {result}\n\n")

print("Extracted Tags:")
for tag in result.tags:
    print(tag)
```

Listing 1: Color mixing program

```
GIM Query: In additive RGB color mixing (light/screen model), mixing red (<|MASKED desc="a hex color code" regex="#[0-9a-fA-F]{6}"|><|/MASKED|>) and green (<|MASKED desc="a hex color code" regex="#[0-9a-fA-F]{6}"|><|/MASKED|>) results in the color <|MASKED desc="a standard color name"|><|/MASKED|> (<|MASKED desc="a hex color code" regex="#[0-9a-fA-F]{6}"|><|/MASKED|>).


GIM Result: In additive RGB color mixing (light/screen model), mixing red (#FF0000) and green (#00FF00) results in the color yellow (#FFFF00).


Extracted Tags:
<|MASKED id="m_0" desc="a hex color code" regex="#[0-9a-fA-F]{6}"|>#FF0000<|/MASKED|>
<|MASKED id="m_1" desc="a hex color code" regex="#[0-9a-fA-F]{6}"|>#00FF00<|/MASKED|>
<|MASKED id="m_2" desc="a standard color name"|>yellow<|/MASKED|>
<|MASKED id="m_3" desc="a hex color code" regex="#[0-9a-fA-F]{6}"|>#FFFF00<|/MASKED|>
```

Listing 2: Color mixing program output

B SQL Perspective on GIM

The Constrained MAP optimization (Eq. 3) can be interpreted through a database query lens. Listing 3 presents a pseudo-SQL simulation of GIM for CV extraction, where the LLM's latent space is queried as a probabilistic knowledge store with semantic and structural constraints.

```sql
SELECT * FROM   LLM_Latent_Space
WHERE  Context = "{CV}\nName: <m1>, Age: <m2>"
-- Semantic Guidance
  AND  m1 MATCHES REGEX "[A-Z][a-z]+"
  -- Structural Guidance
  AND  m2 MATCHES REGEX "\d{2}"
  -- Structural Guidance
ORDER BY Posterior_Probability DESC
-- MAP Optimization
LIMIT  1
```

Listing 3: Pseudo-SQL simulation of GIM for CV extraction.

C Constraint Compilation Details

This appendix details the core logic of the compiler for translating GIM Queries into CFG and JSON Schema.

C.1 CFG Compilation Logic

For models supporting logit intervention, the process of constructing the `llguidance` grammar involves three main steps:

1. **Global Topology Construction:** The compiler first scans all tokens in the query to build a flattened global start rule. This rule strictly defines the sequence of static text and Masked Slots. To tolerate potential whitespace output by the model between fields, we insert `REGEX` (Whitespace) rules between slots. This establishes the generation skeleton, forcing sequential filling and preventing skipping or reordering.

2. **Terminal Reuse Optimization:** This is a critical step for optimizing inference overhead.

The compiler maintains a symbol table based on regex hash values. When traversing each slot, the algorithm checks if the defined Regex exists in the table:

- **Hit:** Directly retrieve the existing terminal ID (e.g., T_k) without creating a new state machine.
- **Miss:** Create a new terminal rule for the regex and register it in the table.

By this method, the underlying inference engine maintains only one regex state regardless of how many identically formatted "date" fields exist.

3. **Slot Rule Definition:** Finally, the compiler generates non-terminal rules for each slot. Here, we bind specific terminals (T_k) to GIM protocol tags. Specifically, we utilize the `suffix` attribute to introduce a **Lookahead Constraint**: forcing the model to generate the closing tag `<|/MASKED|>` immediately after satisfying the regex. This effectively resolves the "halting problem" in regex matching, preventing content overflow.

C.2 JSON Schema Transpilation

For closed-source models offering only API access, the compiler executes a fallback strategy, mapping the GIM object model to JSON Schema. The mapping logic treats the GIM Query as a flattened object, converting all Masked Tags into top-level properties with keys named `m_{id}`.

Specific property mapping rules are as follows:

- **Regex Constraints:** GIM `regex` attributes are converted to JSON Schema `pattern` fields. The compiler automatically wraps anchors `^...$` to ensure strict full matching.
- **Description:** GIM `desc` maps directly to the `description` field.
- **Anti-hallucination Mechanism:** To prevent generation of undefined fields, we explicitly set `additionalProperties` to `False` and mark all defined slots as `Required`.

The transpiled JSON Schema result corresponding to the query in the main text is shown in Listing 4.

```
{
  "type": "object",
  "properties": {
    "m_0": {
      "type": "string",
      "pattern": "^(#[0-9a-fA-F]{6})$",
      "description": "a hex color code"
    },
    "m_1": {
      "type": "string",
      "pattern": "^(#[0-9a-fA-F]{6})$",
      "description": "a hex color code"
    },
    "m_2": {
      "type": "string",
      "description": "a standard color name"
    },
    "m_3": {
      "type": "string",
      "pattern": "^(#[0-9a-fA-F]{6})$",
      "description": "a hex color code"
    }
  },
  "required": [
    "m_0",
    "m_1",
    "m_2",
    "m_3"
  ],
  "additionalProperties": false
}
```

Listing 4: Transpiled JSON Schema Example

D System Prompt and Demonstrations

The system prompt and demonstrations used for Inference-Time Adaptation are shown in Listing 5.

```
SYSTEM_PROMPT_MSG = {
"role": "system",
"content": (
"You complete masked spans. Follow these strict rules:\n"
"1. Input is wrapped in `<|GIM_QUERY|> ... <|/GIM_QUERY|>`.\n"
"2. You must output exactly one `<|GIM_RESPONSE|> ... <|/GIM_RESPONSE|>`.\n"
'3. For every <|MASKED id="m_X" ...|><|/MASKED|> in the query:\n'
'     - Output <|MASKED id="m_X"|> FILLED_TEXT<|/MASKED|>.\n'
"     - Keep id the same.\n"
"     - Remove desc and any other attributes.\n"
"4. Do not add, remove, reorder, or modify any tags.\n"
"5. No explanations. Output only the response."
),
}

DEMO_CONVERSATION_MSGS = [
{
"role": "user",
"content": '<|GIM_QUERY|>Hello, <|MASKED id="m_0" desc="a simple phrase"|><|/MASKED|>.<|/GIM_QUERY|>',
},
{
```

```
"role": "assistant",
"content": '<|GIM_RESPONSE|><|MASKED id="m_0"|>nice to meet you<|/MASKED|><|/GIM_RESPONSE|>',
},
{
"role": "user",
"content": '<|GIM_QUERY|><|MASKED id="m_0" desc="a number"|><|/MASKED|> + <|MASKED id="m_1" desc="a number"|><|/MASKED|> = <|MASKED id="m_2" desc="a number"|><|/MASKED|><|/GIM_QUERY|>',
},
{
"role": "assistant",
"content": '<|GIM_RESPONSE|><|MASKED id="m_0"|>1<|/MASKED|><|MASKED id="m_1"|>1<|/MASKED|><|MASKED id="m_2"|>2<|/MASKED|><|/GIM_RESPONSE|>',
},
# Two more demonstrations omitted...
]
```

Listing 5: System Prompt and Demonstrations

Figure 5: Comparison of CV parsing templates: w/ GIM vs. w/o GIM. Blue boxes indicate schema constraints, while purple boxes indicate context constraints. Note that the Prompt + JSON Schema approach physically separates context and schema definitions.

E Experimental Models

In Table 3, we list the LLMs included in our evaluation.

F Template Comparison for CV Parsing

Figure 5 illustrates the templates used for the CV parsing task with and without GIM.

G Further Experiments

G.1 Code Infilling: HumanEval-Infilling

To assess GIM on deeply structured, syntactically sensitive infilling, we benchmark on **HumanEval-Infilling** (Bavarian et al., 2022). This task requires filling a missing code span such that the completed program passes unit tests; even minor deviations (wrong indentation, spurious newlines) cause failures, making it an extremely strict structural adherence test.

As shown in Table 4, GIM-trained models substantially outperform their base counterparts across both scales (`GIM-1.7B`: +17.5 pp; `GIM-4B`: +20.5 pp), demonstrating that GIM's infilling paradigm generalizes beyond natural language to complex code structures.

G.2 Structured Reasoning Trajectories Mining

We explore whether GIM can directly mine structured, step-aligned reasoning trajectories for Process Supervision (Luo et al., 2024), which typically relies on a fragile "generate CoT then extract steps" pipeline.

Experimental Setup We utilize QASC (Khot et al., 2020) and MedMCQA (Pal et al., 2022) benchmarks to compare three reasoning paradigms:

- **Standard CoT (Baseline):** Uses the standard "Let's think step by step" prompt for free-form CoT.
- **Fixed Budget (GIM):** Pre-allocates a fixed number of masked slots (e.g., $N = 1, 3, 5, 7$) in the query, forcing reasoning within limited steps to probe the model's capability upper bound.
- **Auto Budget (GIM):** Adopts a two-stage filling strategy. A dedicated slot first queries "how many steps are needed?", followed by dynamically expanding the corresponding number of masked slots. This is our target method for trajectory mining.

Results and Analysis Figure 6 validates the effectiveness of GIM in structured mining on QASC and MedMCQA datasets.

Structure without Degradation While GIM is not designed to beat SOTA, experiments show that

Model Name	Parameters
google/gemini-3-flash-preview (Team et al., 2025)	N/A
openai/gpt-5.2 (OpenAI, 2026)	N/A
anthropic/claude-haiku-4.5 (Anthropic, 2025)	N/A
x-ai/grok-4-fast (xAI, 2025)	N/A
deepseek/deepseek-chat-v3-0324 (DeepSeek-AI et al., 2025)	671B
xiaomi/mimo-v2-flash (Team et al., 2026a)	310B
minimax/minimax-m2.1 (MiniMax, 2025)	229B
moonshotai/kimi-k2-thinking (Team et al., 2026b)	170B
qwen/qwen3-30b-a3b-instruct-2507 (Yang et al., 2025)	30B
google/gemma-3-27b-it (Team et al., 2025)	27B
mistralai/ministral-14b-2512 (Liu et al., 2026)	14B
google/gemma-3-12b-it (Team et al., 2025)	12B
Qwen/Qwen3-1.7B (Yang et al., 2025)	1.7B
Qwen/Qwen3-4B-Instruct-2507 (Yang et al., 2025)	4B
GIM-1.7B (ours)	1.7B
GIM-4B (ours)	4B

Table 3: LLMs included in the evaluation. accessed via OpenRouter, accessed via vLLM (Kwon et al., 2023). denotes decoding via JSON Schema, denotes decoding via CFG. The Qwen 1.7B/4B models support both methods.

Model	Acc. (pass@1) ↑
Qwen3-1.7B	4.0%
GIM-1.7B	**21.5%**
Qwen3-4B	24.0%
GIM-4B	**44.5%**

Table 4: HumanEval-Infilling accuracy (zero-shot, pass@1).

imposing strict structure does not systematically impair reasoning. For example, gemma-3-12b-it achieves 80.34% accuracy on QASC in GIM mode, comparable to or slightly better than free CoT (79.59%). This implies we can obtain highly structured data while preserving intelligence. Furthermore, achieving this via supervised learning suggests that future integration with Reinforcement Learning could maintain intelligence even more effectively.

Effectiveness of Auto Budget The Auto Budget strategy demonstrates excellent adaptability. Qwen3-4B achieves 83.02% accuracy in Auto mode, approaching the best-tuned Fixed Budget result. This indicates the model possesses metacognition regarding its reasoning complexity, which GIM successfully explicates into a mineable control signal.

Implications for Data Mining For constructing Process Reward Model (PRM) training data, GIM offers a robust alternative to "post-hoc parsing." GIM enables the one-step generation of well-aligned positive trajectories. Additionally, its in-place rewriting capability allows for efficient synthesis of negative samples by fixing certain steps and modifying others. This "Query-based Generation" transforms trajectory construction into a controllable mining process.

G.3 Relation Extraction on SciERC

To evaluate GIM beyond flat extraction schemas, we conduct experiments on **SciERC** (Luan et al., 2018), a challenging benchmark for joint entity and relation extraction in scientific text. SciERC requires simultaneous extraction of entities and multi-hop relations in a complex graph-structured schema—a significantly harder task than flat CV field extraction.

Strict F1 Metric. Standard IE evaluation skips unparsable outputs, creating a *survivor bias*: models with high format-error rates are evaluated only on their simpler, successfully parsed documents, which inflates apparent F1. We propose **Strict F1**: any document with a format error contributes 0 true positives, and all its ground-truth relations count

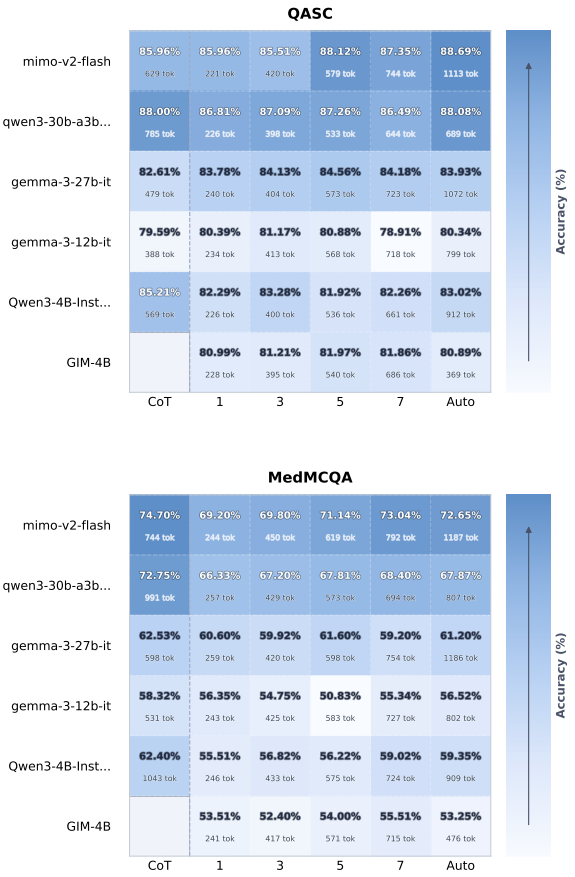

Figure 6: Accuracy heatmap on QASC and MedMCQA benchmarks. Columns correspond to standard CoT, fixed budget and auto budget. Token annotations ("tok") report the mean response tokens.

as false negatives. Strict F1 uses TP normalized against the *full* set of 901 ground-truth relations as the recall denominator, regardless of which documents were successfully parsed.

Model	Error Rate	Standard F1	Strict F1
Qwen3-1.7B	59%	6.46%	3.36%
Qwen3-4B	1%	4.82%	4.77%
GIM-1.7B	10%	1.23%	1.13%
GIM-4B	**0%**	**5.42%**	**5.42%**

Table 5: Relation extraction on SciERC test split. Strict F1 penalizes format errors against the full 901 ground-truth relations.

Analysis. SciERC is inherently difficult for models at this scale; all models achieve low absolute F1, reflecting the complexity of nested graph-structured schemas. The key finding is **format stability**: `GIM-4B` is the only model with a **0% format error rate**, so its Standard and Strict F1 are identical (5.42%, highest overall). In contrast, `Qwen3-1.7B`'s seemingly higher Standard F1 (6.46%) collapses to 3.36% under strict evaluation, exposing the survivor bias—it was only scored on the 41 simplest documents (357 of 901 gold relations) while silently failing on the other 59.

G.4 Decoding Throughput Analysis

We report wall-clock throughput and Time-To-First-Token (TTFT) for `Qwen3-1.7B` and `GIM-1.7B` across three decoding strategies on identical hardware.

Setting	Pred. Rate	Match Rate	Thpt. (tok/s)	TTFT (s)
w/o constrained decoding				
Qwen3-1.7B	74.47%	16.67%	63.98	0.116
GIM-1.7B	83.26%	13.88%	354.14	0.068
w/ constrained decoding in JSON				
Qwen3-1.7B	90.50%	72.00%	298.85	0.074
GIM-1.7B	92.52%	75.10%	330.83	0.073
w/ constrained decoding in GIM format				
Qwen3-1.7B	91.12%	73.15%	138.51	0.119
GIM-1.7B	92.91%	74.90%	145.48	0.116

Table 6: Throughput comparison across decoding strategies.

Within each constrained decoding setting, `GIM-1.7B` consistently achieves higher accuracy and throughput than `Qwen3-1.7B`, attributable to the internalized GIM prior requiring fewer token-level corrections. CFG decoding introduces overhead (138–145 tok/s) from FSM state maintenance; this gap is expected to narrow with future engineering optimizations (tokenizer redesign, quantization). Note that GIM models are designed to be used *with* constrained decoding; the unconstrained (w/o c.s.) row is included for completeness.

G.5 XML Representation Ablation

A key question is whether GIM's gains stem from the XML-based neuro-symbolic representation itself or simply from additional fine-tuning. To isolate these effects, we train an **AR-1.7B** baseline on the identical dataset formatted as standard instruction-following QA pairs without any GIM Query/Response structure. As shown in Table 7, removing the XML structure consistently degrades all three metrics—most notably reducing Match Rate by 3.59 percentage points—confirming that the neuro-symbolic representation is itself a critical contributor, independent of fine-tuning.

Model	WNP ↓	Pred. Rate ↑	Match Rate ↑
AR-1.7B	39.52%	88.24%	73.45%
GIM-1.7B	**37.47%**	**93.87%**	**77.04%**

Table 7: Ablation on XML structure: standard QA fine-tuning (AR-1.7B) vs. GIM-structured fine-tuning (GIM-1.7B) on identical training data.

G.6 Training Data Ablation Studies

G.6.1 Masking Strategy: Poisson vs. Semantic Masking

We train models with different Poisson-to-Semantic masking ratios to quantify each strategy's contribution. As shown in Table 8, the balanced 50%/50% split outperforms pure Poisson masking on both metrics, confirming that semantic and reasoning-focused masking is essential for GIM to generalize to real-world extraction.

Poisson : Semantic	Pred. Rate ↑	Match Rate ↑
50% : 50%	**71.11%**	**62.87%**
100% : 0%	57.19%	54.09%

Table 8: Effect of masking ratio on `GIMBench-regex-match`.

G.6.2 Impact of High-Quality Reasoning Data

To validate the importance of high-quality CoT and instruction subsets (LIMA, HK-O1aw, GSM8k, etc.), we train a variant removing all such subsets and retaining only 50k mid/low-quality samples. On the CV parsing task, accuracy drops from **66.31%** (full recipe, `GIM-4B`) to **43.26%**—an absolute reduction of over 22 percentage points. This confirms that GIM's structural adherence in complex real-world tasks is anchored in high-quality reasoning data; without these subsets, the model's structural attention mechanism collapses under real-world extraction conditions.

H Computational Resources

We performed all training and evaluation on a multi-GPU workstation. The node provides **12× NVIDIA RTX 4090D GPUs** (24 GB each), paired with a **16-core Intel Xeon Platinum 8481C** CPU and **80 GB RAM**. The system includes a **30 GB OS disk** and a **50 GB data disk**, expandable to **2.5 TB**.