Extraction stopping calibration

A probabilistic approach to graph RAG efficiency through novel yield ratio modeling

Emilio Gagliardi August 2026 Technical Report

---

1. How graph RAG systems extract knowledge

The dominant paradigm in graph-based retrieval-augmented generation is exhaustive extraction: given a text chunk, the LLM extracts every entity, relation, and claim it can find. Microsoft's GraphRAG implementation exemplifies this through a **gleaning** mechanism - multi-round continuation prompts where the LLM assesses whether all entities have been extracted. If the model detects missing information, a `CONTINUE_PROMPT` triggers additional passes. A `LOOP_PROMPT` decides when to stop, but the objective is completeness, not efficiency.

This approach is expensive. A single knowledge graph extraction run with GPT-4o-mini has been reported at 27 hours and $3,500; a follow-up with GPT-4.1 exceeded $14,000. Graph extraction constitutes roughly 75% of GraphRAG's total indexing cost. These numbers have driven a wave of efficiency-focused research, but the solutions attack the problem from a different direction than the one described in this report:

System	Approach	Tradeoff
TERAG (2025)	Token-efficient graph construction	80% accuracy at 3–11% of token cost, but accepts lower quality
E²GraphRAG (2025)	SpaCy for structural pass, LLM only for summaries	~10x speedup, but loses LLM-quality entity extraction

FastGraphRAG	Cheaper extraction pipeline	Noisier graphs - the tradeoff is accepted, not managed
SocraticKG (2026)	QA-driven fact extraction via 5W1H questioning	Better coverage, but higher cost per section

The closest academic work to the approach described here is Giordano & Razniewski's "Foundations of LLM Knowledge Materialization" (EACL 2026), which studies whether recursive LLM-based extraction can even terminate. They measure yield decay across iterations but treat convergence as a theoretical property to observe, not a production criterion to calibrate.

> **The gap:** Every system above either extracts everything and filters later, or reduces per-pass cost. None of them model diminishing returns within a section and fit a stopping criterion. The extraction stopping calibration described below is, as far as current literature indicates, a novel approach.

---

2. The extraction pipeline

The ETL pipeline runs multi-pass LLM extraction on each document section. After each pass, it computes a novel yield ratio and evaluates it against a calibrated logistic regression model to decide whether another pass is worth the cost. The pipeline loops until the model predicts that additional extraction is unlikely to surface new valuable claims.

FIGURE 1 - PIPELINE FLOW ACROSS THREE EXTRACTION PASSES

	APP SENDS	LLM EXTRACTS	COMPUTER	EVALUATE	DECISION
Pass 1	**Section text** `no prior context`	**$k_1 = 3$ clauses** `first extraction`	**$r_1 = 1.00$** `3 new / 3 total`	**P(cov) = 0.00** `$r_1 = 1.00 > T$`	**Continue** `→ loop to pass 2`
Pass 2	**Section + 3 known**	**$k_2 = 2$ clauses**	**$r_2 = 0.40$**	**P(cov) = 0.17**	**Continue**

	C_1 = 3 as ignore list	2 novel finds	2 new / 5 total	r_2 = 0.40 > T	→ loop to pass 3
Pass 3	**Section + 5 known** C_2 = 5 as ignore list	**k_3 = 0 clauses** nothing new	**r_3 = 0.00** 0 new / 5 total	**P(cov) = 0.83** r_3 = 0.00 < T	**Flag complete** extraction_complete
Pass N	**Section + C_{n-1} known** all prior clauses as ignore list	**k_n = ? clauses** novel finds this pass	**$r_n = k_n / C_n$** novel / cumulative	**P(cov | r_n)** $\sigma(\beta_0 + \beta_1 \cdot r_n)$	**$r_n \gtrless T$?** stop or continue

Each row is one LLM pass. The app sends the section plus any previously extracted clauses as an ignore list. The LLM extracts novel clauses. The novel yield ratio r is computed and evaluated against the calibrated stopping sigmoid. When r drops below the threshold T, the section is flagged as fully extracted.

---

3. How the model is fitted: MLE and Nelder-Mead

Before defining the stopping model's variables, it is worth understanding the two ideas that underpin how its coefficients are found: **maximum likelihood estimation** (the objective) and the **Nelder-Mead simplex algorithm** (the optimizer). Together they answer the question: given a set of observed extraction outcomes, what values of β_0 and β_1 make those outcomes most probable?

3.1 Maximum likelihood estimation

Each annotated section produces a binary observation after an extraction pass: either coverage has been achieved (y = 1) or it has not (y = 0). The logistic model predicts the probability of coverage as a function of the novel yield ratio:

$$P(y = 1 \mid r) = \sigma(\beta_0 + \beta_1 \cdot r) \quad \text{where} \quad \sigma(z) = 1 / (1 + e^{-z})$$

For a dataset of m observations $\{(r^{(i)}, y^{(i)})\}$, the **likelihood** is the probability of seeing all the observed labels given the model's parameters. Because each observation is independent, the joint probability is the product of individual probabilities:

$$L(\beta_0, \beta_1) = \prod_{i=1..m} p_i^{y(i)} \cdot (1 - p_i)^{1-y(i)}$$
$$\text{where } p_i = \sigma(\beta_0 + \beta_1 \cdot r^{(i)})$$

Maximizing this product directly is numerically unstable (multiplying many small probabilities underflows to zero), so we take the logarithm. Since log is monotonically increasing, the parameters that maximize L also maximize log L:

$$\ell(\beta_0, \beta_1) = \Sigma_{i=1..m} [\, y^{(i)} \cdot \log(p_i) + (1 - y^{(i)}) \cdot \log(1 - p_i) \,]$$

This is the **log-likelihood** - and its negation is exactly the **cross-entropy loss** familiar from machine learning. Maximizing the log-likelihood is mathematically identical to minimizing cross-entropy. The connection is not a coincidence: cross-entropy measures how far the model's predicted distribution is from the observed distribution, and MLE finds the parameters that close that gap.

For a single-feature logistic regression (one input r, two parameters β_0 and β_1), the log-likelihood surface is concave - it has exactly one global maximum and no local maxima. This means any optimizer that climbs the surface will eventually find the optimal coefficients. The question is which optimizer to use.

3.2 The optimizer landscape

Logistic regression has a well-studied optimization problem, and several families of algorithms compete for the role. Minka (2003) benchmarked

eight optimizers head-to-head; Pedregosa (2013) extended this comparison using scipy implementations. The table below summarizes the viable options, organized by how much curvature information each method uses.

FIGURE 1A - OPTIMIZER COMPARISON FOR LOGISTIC REGRESSION MLE

Optimizer	Order	Per-iteration cost	Convergence	Key property
Newton-Raphson / IRLS	2nd	$O(d^2n + d^3)$	Quadratic	Exact Hessian. The standard for GLMs. Converges in 4-10 iterations. Equivalent to iteratively reweighted least squares (IRLS) for logistic regression.
Trust Region Newton	2nd	$O(d^2n + d^3)$	Quadratic	Robust to ill-conditioned designs. Dominates Pedregosa's benchmarks across all correlation levels. Used by LIBLINEAR.
L-BFGS	Quasi-2nd	$O(nd + md)$	Superlinear	Approximates Hessian from gradient history (m recent vectors). Memory-efficient at high d. scikit-learn's default solver.
BFGS	Quasi-2nd	$O(d^2 + nd)$	Superlinear	Stores full approximate Hessian. At d = 2, identical to L-BFGS in

				practice. No memory advantage needed.
Conjugate Gradient	1st	`O(nd)`	Linear	No Hessian storage. Competitive with quasi-Newton in Minka's benchmarks. Degrades with correlated features.
Böhning Fixed-Hessian	2nd (approx)	`O(nd)`	Linear	Pre-computes a fixed lower-bound Hessian $\bar{H} = -\frac{1}{4}X^TX$ once. Guaranteed monotone convergence. More iterations but each is cheap.
Nelder-Mead	0th	`O(n)` per vertex	Sublinear	Derivative-free. Uses only function evaluations. Robust to noisy objectives. No gradient or Hessian required.

d = number of parameters, n = number of observations, m = memory depth for L-BFGS. "Order" refers to the highest derivative the optimizer uses: 2nd-order methods use the Hessian, quasi-2nd approximate it, 1st-order methods use only the gradient, and 0th-order methods use only function values.

For large-scale logistic regression (d in the thousands, n in the millions), these differences matter enormously. Trust Region Newton and L-BFGS dominate production systems: scikit-learn defaults to L-BFGS, LIBLINEAR uses Trust Region, and statsmodels offers Newton-CG and IRLS. Nelder-Mead and BFGS would be impractical at that scale.

But the stopping calibration operates at d = 2 parameters and n ≈ 450 observations. At this scale, every optimizer in the table converges in under a

millisecond. The 2×2 Hessian is trivially cheap to compute and invert; L-BFGS has no memory advantage over full BFGS; and the per-iteration cost differences between O(nd) and O(n) are negligible. The differentiator is not speed but **robustness and simplicity** - which optimizer is most reliable on a small, noisy dataset with no hyperparameter tuning.

3.3 The Nelder-Mead simplex algorithm

Nelder-Mead is a **derivative-free** optimizer that navigates the parameter space using only function evaluations - no gradients, no Hessians.

The algorithm works by maintaining a geometric shape called a **simplex** in parameter space. For our two parameters (β_0, β_1), the simplex is a triangle - three candidate solutions forming a shape on the 2D parameter plane. At each iteration, the algorithm evaluates the log-likelihood at each vertex, identifies the worst vertex, and attempts to replace it with a better point using four possible moves:

FIGURE 1B - NELDER-MEAD SIMPLEX OPERATIONS

REFLECT

Mirror the worst vertex through the centroid of the remaining vertices. If the reflected point is better than the second-worst but not better than the best, accept it.

EXPAND

If the reflected point is the new best, push further in that direction. The simplex stretches toward the optimum, taking larger steps when it's making progress.

CONTRACT

If the reflected point is still the worst, pull back toward the centroid. The simplex tightens around a promising region, taking smaller steps near the optimum.

SHRINK

If nothing helps, shrink the entire simplex toward the best vertex. All vertices except the best move closer to it. A last resort that narrows the search.

The simplex starts as a triangle in (β_0, β_1) space. Through repeated reflect/expand/contract/shrink operations, it crawls across the log-likelihood surface until it converges on the maximum - the optimal coefficients.

3.4 Why Nelder-Mead for this problem

Given that every optimizer in Figure 1a converges to the same answer at d = 2, the choice comes down to practical considerations at n ≈ 450 with noisy annotation data. In this regime, Nelder-Mead has three advantages over the gradient-based alternatives:

No derivative estimation. Newton-Raphson and L-BFGS require the gradient of the log-likelihood (and Newton requires the Hessian). For logistic regression the gradient has an analytic form, so this is not a fundamental barrier - but it is additional code to implement and validate. Nelder-Mead sidesteps derivatives entirely, evaluating the log-likelihood directly at each candidate point. With only two parameters, the cost of extra function evaluations is negligible.

Robustness to noisy objectives. The annotation data contains measurement noise (the LLM's claim count is an estimate, not an exact count). Gradient-based methods follow a single derivative direction that noise can distort. Nelder-Mead's simplex geometry naturally smooths over local irregularities - it compares function values at multiple points rather than chasing a noisy gradient. This matters more here than in typical logistic regression because the "labels" (coverage achieved / not achieved) are themselves uncertain: they depend on the LLM's extraction quality, which varies by prompt and model version.

Simplicity of implementation. The fitting code in `calibration_fit.py` is 162 lines. It calls `scipy.optimize.minimize(method='Nelder-Mead')` on the negative log-likelihood (because scipy minimizes, and we want to maximize). The entire optimization is one function call with no hyperparameter tuning - no learning rate, no batch size, no convergence schedule. Switching to L-BFGS or Newton-CG would require supplying the gradient (and optionally Hessian) callbacks, adding validation logic, and gaining no measurable improvement in fit quality at this scale.

> **When to reconsider:** If the stopping model is later enriched with additional covariates (Section 6.5 proposes adding sentence count, document type, and pass number, raising d from 2 to 4-5), the argument for Nelder-Mead weakens. At d = 5, the simplex has 6 vertices and

convergence slows relative to gradient-based methods. The natural upgrade path is `scipy.optimize.minimize(method='L-BFGS-B')` with an analytic gradient - a one-line solver change plus a gradient function.

The result: Nelder-Mead returns the β_0 and β_1 that maximize the probability of the observed coverage outcomes given the observed yield ratios. The decision boundary $T = -\beta_0/\beta_1$ falls out algebraically - it is the ratio value where P(coverage) = 0.5, the point of maximum uncertainty about whether the section is exhausted.

---

4. The novel yield ratio and stopping sigmoid

4.1 Variable definitions

All random variables and parameters must be defined before the model is specified. The stopping calibration operates on a single feature - the novel yield ratio - and produces a single output: a coverage probability.

SYMBOL	DEFINITION
N	Total clauses in a section (here N = 8 in the worked example)
V	Count of truly valuable clauses in the section (unknown in production; here V = 5)
t	Extraction pass number (t = 1, 2, 3, ...)
k_t	Novel clauses extracted on pass t (decided by the LLM, not the model)
C_t	Cumulative clauses after pass t: $C_t = C_{t-1} + k_t$
r_t	Novel yield ratio: $r_t = k_t / C_t$ (fraction of cumulative that is new this pass)
β_0, β_1	Logistic regression coefficients, calibrated offline via MLE (Nelder-Mead) on historical extraction data

`P(cov | r)`	Probability the section is fully extracted: $P = \sigma(\beta_0 + \beta_1 \cdot r)$ where σ is the sigmoid function
`T`	Decision boundary: $T = -\beta_0 / \beta_1$. When $r_t < T \rightarrow P > 0.5 \rightarrow$ stop extracting

4.2 A hypothetical section

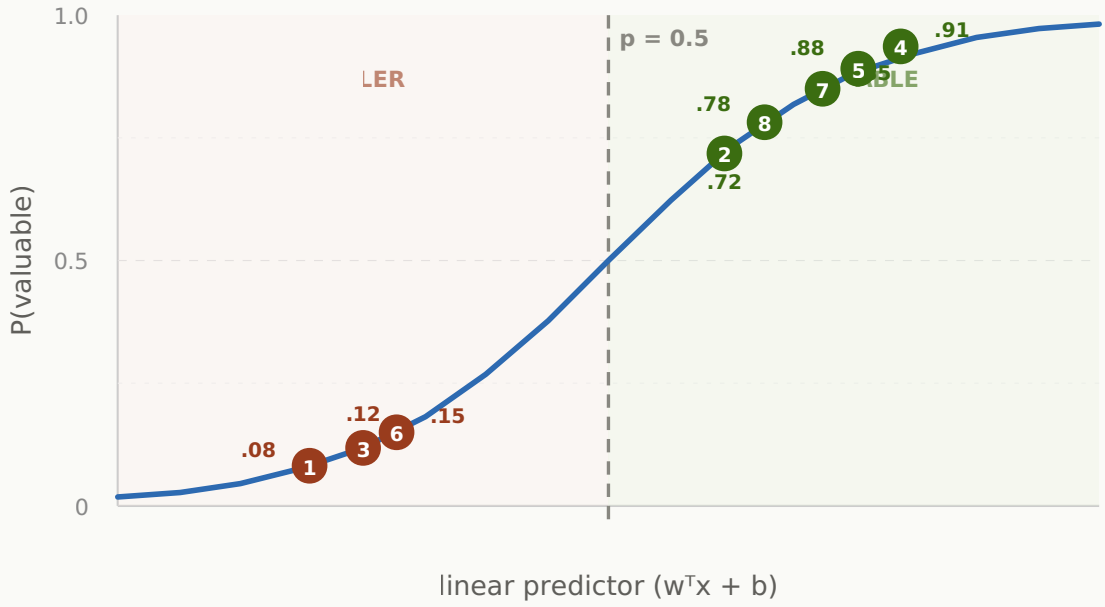
Consider an 8-sentence section from a master services agreement. Five sentences contain substantive contractual obligations (valuable clauses); three are boilerplate, preamble, or cross-references (filler). The LLM does not know this classification in advance - it discovers valuable content through extraction.

FIGURE 2 - 8-CLAUSE SECTION WITH CLASSIFICATION PROBABILITIES

S1	This Agreement may be referred to as the 'Master Services Agreement' dated as of the Effective Date.	0.08	filler
S2	Provider shall deliver all Services in accordance with the service levels set forth in Schedule B and shall promptly notify Client of any anticipated failure to meet such levels.	0.72	valuable
S3	The following definitions apply throughout this Agreement unless the context otherwise requires.	0.12	filler
S4	Neither party shall assign or transfer any rights or obligations under this Agreement without the prior written consent of the other party.	0.91	valuable
S5	Provider's aggregate liability under this Agreement shall not exceed the total fees paid by Client during the twelve-month period preceding the claim.	0.88	valuable
S6	The provisions of this section shall be read in conjunction with the schedules and exhibits attached hereto.	0.15	filler
S7	Client shall have the right to terminate this Agreement for cause upon thirty days' written notice if Provider fails to cure a material breach within such period.	0.85	valuable
S8	All Confidential Information disclosed by either party shall remain the exclusive property of the disclosing party and shall not be used except as necessary to perform obligations under this Agreement.	0.78	valuable

Each clause's feature vector x (section position, keyword density, syntactic structure, surrounding context) is mapped through a logistic function `p = σ(wᵀx + b)` to produce a probability of being "valuable." The model does not see the ground-truth labels - it scores each sentence independently, and sentences above the 0.5 decision boundary are classified as likely to contain extractable claims.

FIGURE 2A - CLAUSE PROBABILITIES ON THE CLASSIFICATION SIGMOID

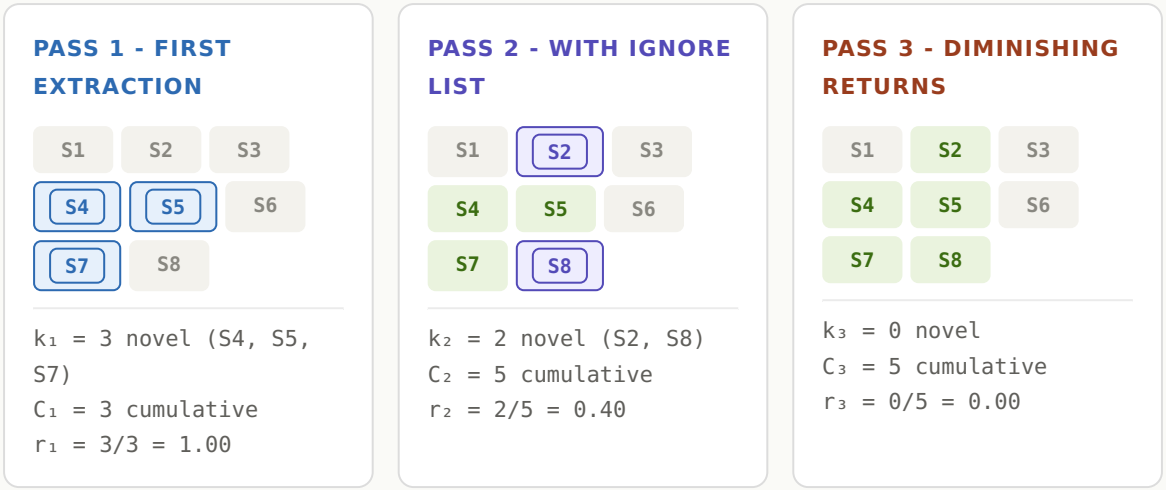


Each numbered dot is one sentence from the section above. The blue curve is σ(z) - the standard logistic function. Filler sentences (S1, S3, S6) cluster at the left where p < 0.5. Valuable clauses (S2, S4, S5, S7, S8) sit to the right where p > 0.5. This per-clause classification is what the LLM performs internally during extraction - the stopping model described in Section 4.4 operates at the section level, not the clause level.

4.3 Extraction across passes

The LLM processes the same section repeatedly, receiving previously extracted clauses as an ignore list on each subsequent pass. On each pass, it finds fewer novel clauses - the most obvious ones surface first, with diminishing returns on later passes.

FIGURE 2B - MULTI-PASS EXTRACTION WALK-THROUGH

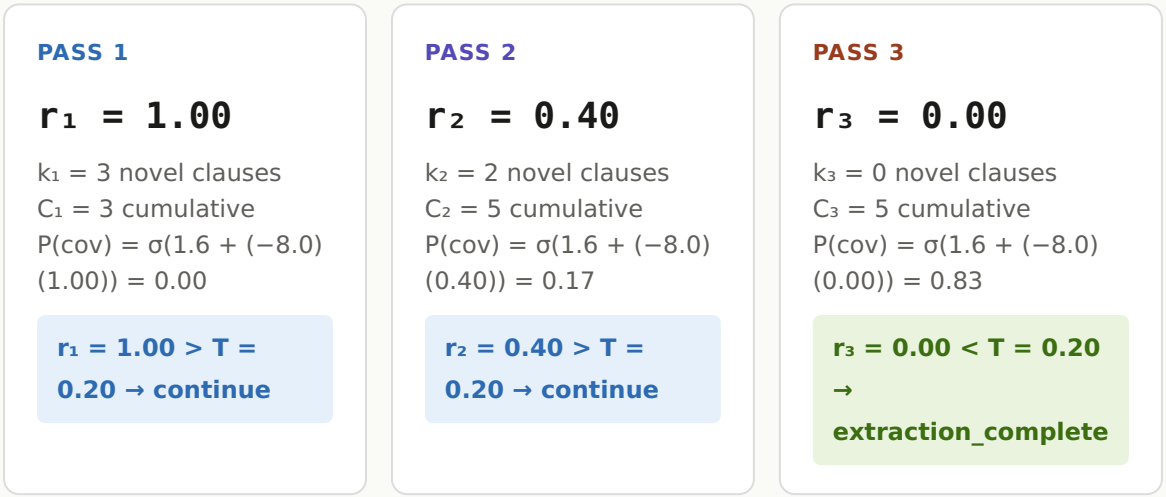


*Colored tags show extraction state per sentence. **Blue** = newly extracted this pass. **Purple** = newly extracted pass 2. **Green** = previously extracted. Gray = not extracted. The LLM finds the most prominent clauses first (S4: assignment restriction, S5: liability cap, S7: termination rights), then catches subtler ones on the second pass (S2: service levels, S8: confidentiality).*

4.4 The stopping evaluation

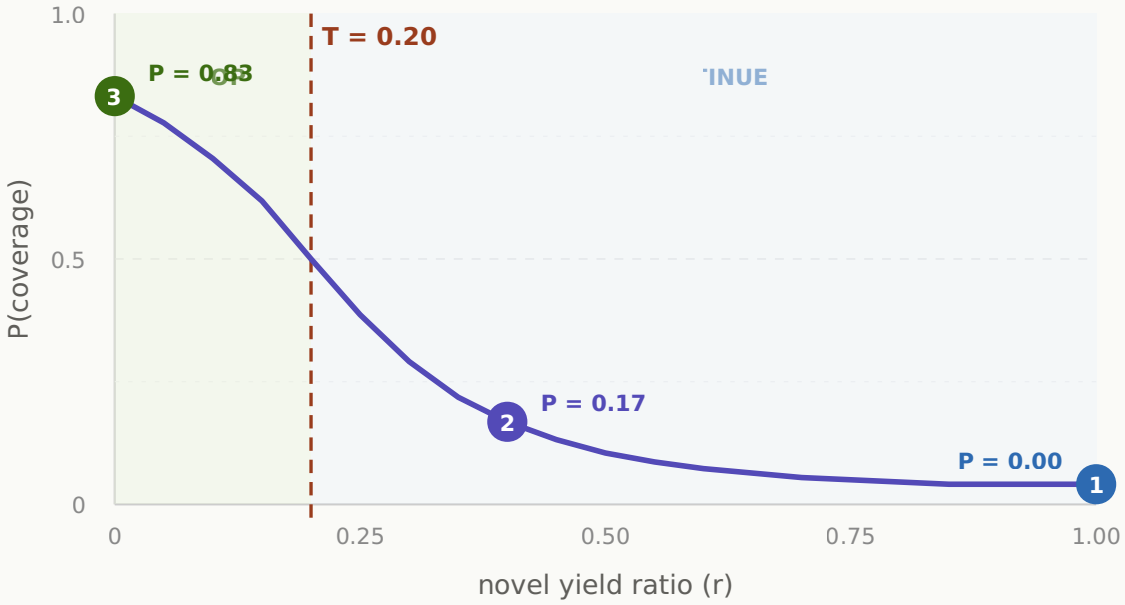
After each pass, the novel yield ratio r is fed to the calibrated logistic regression. The sigmoid maps r to a coverage probability. High ratio means low coverage probability (keep extracting); low ratio means high coverage probability (stop). The decision boundary T is where P(coverage) crosses 0.5.

FIGURE 3 - STOPPING EVALUATION PER PASS



The sigmoid never touches individual clauses. It sees one number - the yield ratio - and outputs a coverage probability. The LLM does all clause extraction; the logistic regression only decides when to stop asking.

FIGURE 3A - THE STOPPING SIGMOID CURVE



The purple curve shows P(coverage | r) = σ(1.6 − 8.0 · r). Pass 1 (r = 1.00) sits at the far right where coverage probability is near zero - all clauses are novel, so the model is confident extraction is incomplete. Pass 2 (r = 0.40) remains in the continue zone. Pass 3 (r = 0.00) jumps to the left of the threshold T = 0.20, where coverage probability (0.83) exceeds 0.5, triggering the extraction_complete flag.

---

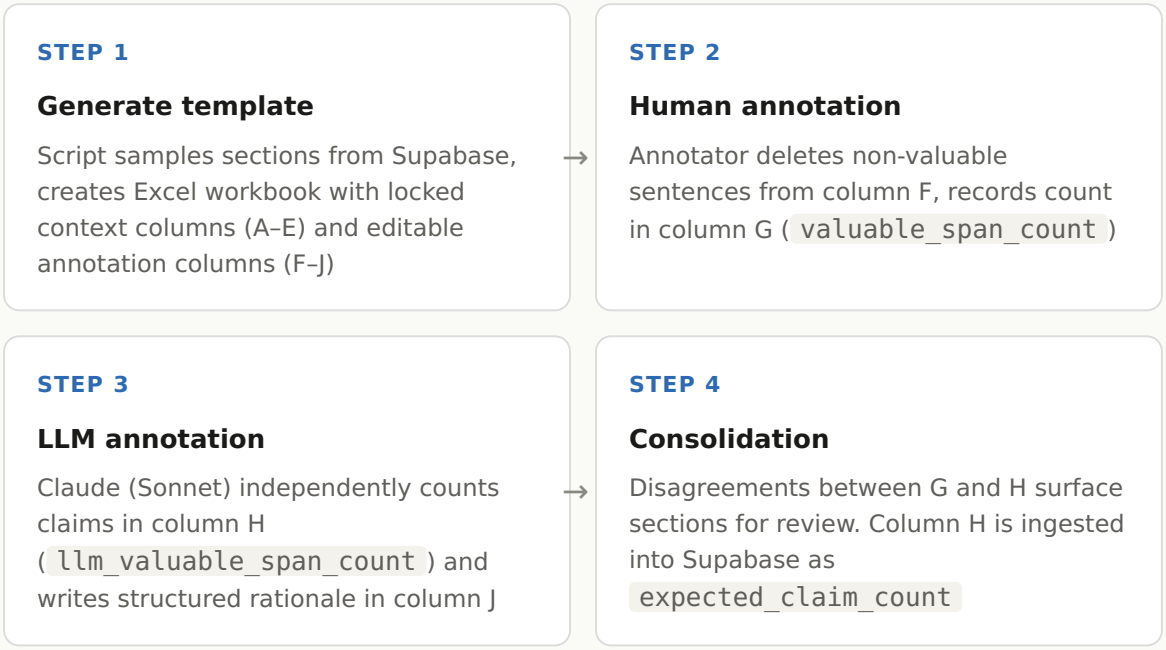
5. Training the stopping model

5.1 The annotation workflow

The stopping model's training data comes from Excel annotation templates generated from live Supabase data. Sections are sampled at ~150 per document category across three categories (e.g., contracts, policies, regulatory filings), yielding ~450 annotated sections total. Sections with known boilerplate type classifications - tables of contents, cover pages, appendix headers - are excluded before sampling, so the dataset contains only sections where clause extraction is meaningful. Within each category,

sections are drawn from ~20 documents at 8-12 sections per document. Two annotators work each section independently:

FIGURE 4 - ANNOTATION PIPELINE



5.2 Why column H, not column G

A natural question: the human annotator (column G) consistently undercounts relative to the LLM (column H). Why use the LLM's count as ground truth?

The answer lies in what the stopping model optimizes for. The calibration targets a `target_recall` of 0.85 - it asks, "have we extracted at least 85% of the expected claims?" If the expected count is too low (using the conservative human count), the model would declare coverage satisfied too early, stopping extraction before the LLM has finished finding value. If the expected count is too high (using the LLM's more thorough count), the model errs on the side of over-extraction - running an extra pass costs tokens but doesn't lose information.

> **The systematic undercounting is a feature, not a bug.** By using the LLM's higher claim count as the target, the stopping model is conservative -

it keeps extracting until it's genuinely confident the section is exhausted. This trades a small amount of extra LLM cost for materially higher recall.

5.3 From annotations to training signal

The logistic regression is not trained directly on the annotation spreadsheet. The spreadsheet provides the **expected claim count** per section - the denominator. The numerator comes from production extraction runs. Here is how the two data sources combine:

SOURCE	WHAT IT PROVIDES
`Annotation XLSX (column H)`	`expected_claim_count` - how many extractable claims the LLM judge says the section contains. Ingested into Supabase's `extraction_annotations` table.
`Production ETL (extraction runs)`	`extraction_attempts` - actual yields per pass: how many claims the LLM found on pass 1, pass 2, etc., for each annotated section.

The calibration Dagster asset (`calibrated_stopping_thresholds` , scheduled Sunday 3 AM) joins these two tables and constructs the training data for logistic regression:

Training data construction:
1. For each annotated section, compute `novel_yield_ratio` at each pass from `extraction_attempts`
2. Compute the binary label: `coverage_achieved = 1` if `actual_claims / expected_claims ≥ 0.85` , else `0`
3. Filter out `zero_yield` and `bridge_only` sections (no meaningful calibration signal)
4. Fit logistic regression: `novel_yield_ratio` (X) → `coverage_achieved` (y)
5. The resulting β_0, β_1 define the sigmoid; $T = -\beta_0/\beta_1$ is the decision boundary

The model learns a simple mapping: "when the novel yield ratio is this value, what is the probability that we have already achieved 85% coverage?" A ratio of 1.0 (first pass, everything is new) maps to near-zero

coverage probability. A ratio of 0.0 (nothing new found) maps to high coverage probability. The sigmoid smoothly interpolates between these extremes, and the threshold T identifies the inflection point.

5.4 Model stratification

The annotation dataset is deliberately structured to support this comparison: ~150 sections per category across three document types gives ~450 sections total. The calibration fits both a pooled model (all 450 sections combined) and per-`document_category` stratified models (~150 sections each). A likelihood ratio test (LRT, chi-squared) determines whether stratification provides statistically significant improvement. Different document types may have different extraction saturation patterns - a clause-heavy services agreement saturates differently than a high-level corporate policy. If the lexical or structural differences between document groups are sufficiently divergent and the LRT is significant ($p < 0.05$), per-category thresholds are deployed; otherwise, the pooled threshold is used.

Diagnostics reported per fit include AIC (model selection), Brier score (calibration quality), and log-likelihood (goodness of fit). The fitting uses scipy's Nelder-Mead optimizer for MLE, which is robust to the small sample sizes typical of annotation datasets.

5.5 Runtime inference

At extraction time, `should_stop_section()` reads the calibrated coefficients from `extraction_stopping_thresholds` in Supabase. After each LLM pass, it computes the current novel yield ratio and evaluates it against the sigmoid. When the ratio drops below T for the section's cell (defined by `section_type × strategy × version`), extraction stops.

Sections that never receive the `extraction_complete` flag remain eligible for future re-processing - the system can send them through the pipeline again with the previously extracted claims as an ignore list, potentially surfacing content that was missed due to LLM stochasticity or prompt changes.

6. Alternative distributional models

The logistic regression on yield ratio is a direct model - it maps one number to a binary decision. But the data it operates on has richer statistical structure that the simple sigmoid does not exploit. Sections vary in length (1-15 sentences), valuable clauses cluster within sections, and many sections are trivially exhausted on the first pass. Three alternative distributional models have been proposed to account for these properties.

An important distinction: **Nelder-Mead is the optimizer, not the model.** It can fit any of the models below via MLE. The question is not whether to change the optimizer, but whether to change what is being optimized - the likelihood function itself.

6.1 Beta-Binomial: modeling the clause container

Each section has a finite number of sentences acting as containers. The Beta-Binomial treats the sentence count as a fixed number of trials but allows the probability of finding a valuable clause to vary by section through a Beta prior:

SYMBOL	DEFINITION
n_i	Number of sentences in section i (known at extraction time, 1-15)
Y_i	Number of valuable clauses in section i, where $Y_i \sim \text{BetaBin}(n_i, \alpha, \beta)$
α, β	Shape parameters of the Beta distribution governing clause probability
$E[Y_i]$	Expected clause count: $n_i \cdot \alpha / (\alpha + \beta)$

The Beta distribution introduces an intra-section correlation parameter. When α is large relative to β, most sections are clause-dense. When α and β are both small, clause density varies widely across sections - some are packed with obligations, others are nearly empty. This captures the

"clumping effect": if a section contains one valuable clause, the probability of finding more in that section increases due to shared contractual context.

For the stopping decision, a Beta-Binomial model would predict the expected total clause count $\hat{V}_i = n_i \cdot \alpha/(\alpha+\beta)$ for each section, then compare cumulative extracted clauses C_t against that prediction. The stopping rule becomes: stop when $C_t/\hat{V}_i \geq 0.85$.

> **Where this adds value:** The Beta-Binomial is most useful not as a replacement for the stopping classifier, but as a replacement for the **annotation dependency**. Currently, the expected clause count comes from column H of the annotation spreadsheet. A Beta-Binomial regression on section features (sentence count, word count, document type) could predict that denominator for unannotated sections, enabling deployment to new document types without running the annotation workflow.

6.2 Negative Binomial: modeling the iteration count

The Negative Binomial models overdispersed count data - specifically, the number of discrete LLM passes required to exhaust a section. Where a Poisson model assumes the mean equals the variance (rarely true in text processing), the Negative Binomial adds a dispersion parameter that allows the variance to exceed the mean:

SYMBOL	DEFINITION
N_i	Number of passes until section i is exhausted, $N_i \sim \text{NegBin}(r, p_i)$
r	Dispersion parameter (number of "successes" to reach exhaustion)
p_i	Per-pass exhaustion probability for section i, potentially linked to covariates
$E[N_i]$	Expected pass count: $r(1 - p_i) / p_i$

The stopping rule under this model: compute $P(N_i \leq t)$ after pass t. When this probability exceeds a threshold, stop.

The limitation for this application is that the Negative Binomial treats passes as exchangeable - it models how many passes are needed but ignores what happens within each pass. The novel yield ratio carries strictly more information: it tells you not just that you are on pass 3, but that pass 3 found zero new clauses out of five cumulative. Two sections both on pass 3 might have very different yield ratios, and the logistic regression distinguishes them while the Negative Binomial cannot.

6.3 Zero-Inflated Hurdle: two-stage process

The extraction data has a high concentration of zeros at two levels: sections with nothing to extract (structural zeros) and passes that find nothing new (sampling zeros). A zero-inflated hurdle model splits the process into two stages:

STAGE	MODEL
`Hurdle`	Logistic regression: P(section has extractable content) given section features. Sections failing the hurdle are skipped entirely.
`Count`	Negative Binomial: given the section has content, $N_i \sim \text{NegBin}(r, p_i)$ for the number of passes to exhaust it.

This model has 5-8 parameters (hurdle logistic coefficients plus count model parameters plus covariates). With ~450 annotated sections, the parameter-to-observation ratio becomes thin - particularly for per-document-type fits where each stratum has ~150 sections. Maximum likelihood estimates of dispersion parameters are known to be biased upward with small samples, and confidence intervals exhibit below-nominal coverage in overdispersed regimes.

More practically, the pre-filtering of boilerplate sections (tables of contents, cover pages, appendix headers) already serves as a hurdle. The SetFit section-type classifier assigns each section a type classification before extraction begins, and known non-extractable types are excluded. Adding a statistical hurdle on top of this existing filter is redundant.

6.4 Comparison

FIGURE 5 - MODEL COMPARISON FOR EXTRACTION STOPPING

Model	What it models	Runtime decision	Params	Fit for stopping?
Logistic Regression	Binary: is this section exhausted, given the yield ratio?	`if r < T → stop` needs only the ratio	**2**	**Direct** models the decision boundary itself
Beta-Binomial	Count: how many valuable clauses should this section have?	`if` $C_t/\hat{V} \geq 0.85$ `→ stop` needs sentence count + features	**3-5**	**Indirect** models the denominator, not the decision
Negative Binomial	Count: how many passes to exhaust this section?	`if` $P(N \leq t) > \theta$ `→ stop` needs pass number + features	**3-5**	**Indirect** ignores within-pass yield signal
Zero-Inflated Hurdle	Two-part: has anything to extract? + how many passes?	Stage 1 hurdle + Stage 2 NegBin	**5-8**	**Redundant** pre-filtering already acts as hurdle

All four models can be fitted via MLE using Nelder-Mead. The optimizer is not the differentiator - the model specification is. The logistic regression is the only model that directly produces the stop/continue decision without an intermediate prediction step.

6.5 The recommended path: enriched logistic regression

Rather than replacing the logistic regression with a different distributional family, the more productive direction is to enrich it with the features that the alternative models highlight. The current model uses a single predictor - the novel yield ratio. But the discussion above identifies section-level properties that could shift the decision boundary:

FEATURE	RATIONALE
r_t	Novel yield ratio (current sole predictor). Captures diminishing returns within the extraction sequence.
n_i	Sentence count for section i. Longer sections may have higher clause density and require more passes before the yield ratio becomes informative.
$type_i$	Document category (categorical). Different document types have different clause density distributions - a services agreement saturates differently than a corporate policy.
t	Pass number. The same yield ratio on pass 2 vs pass 5 may carry different stopping implications.

The enriched model becomes:

> **$P(cov \mid r, n, type, t) = \sigma(\beta_0 + \beta_1 r + \beta_2 n + \beta_3 \cdot type + \beta_4 t)$**
> This keeps the direct-decision architecture (a single sigmoid producing a stop/continue probability) while letting section length, document type, and pass number shift the threshold. The existing likelihood ratio test (LRT) determines whether the additional covariates justify their parameter cost.

The fitting pipeline already supports this comparison. The calibration Dagster asset fits both pooled and stratified models and reports AIC, Brier score, and log-likelihood per fit. Adding covariates to the logistic regression is a matter of expanding the feature vector - the Nelder-Mead optimizer handles the additional parameters without modification, and the LRT

provides a principled test of whether the richer model is worth its complexity.

If the LRT shows that sentence count and document type do not significantly improve the fit, the yield ratio alone carries the stopping signal and the current two-parameter model is sufficient. If they do improve it, the enriched model produces per-section thresholds that adapt to structural variation in the corpus.

6.6 Beta-Binomial as a clause count predictor

Independent of the stopping model, the Beta-Binomial has a specific role worth pursuing: predicting expected clause counts for unannotated sections. The current pipeline depends on column H of the annotation spreadsheet to provide the `expected_claim_count` denominator. This creates a bottleneck - every new document type requires annotation before the stopping model can be calibrated.

A Beta-Binomial regression trained on the existing 450 annotated sections could predict $\hat{V}_i = n_i \cdot \alpha/(\alpha+\beta)$ for any new section, given only its sentence count and document type. This would allow deployment to new document categories without running the annotation workflow, with the annotation-derived counts serving as ground truth for validating the Beta-Binomial predictions.

The two models would then operate in sequence: the Beta-Binomial predicts how many clauses a section should contain (the denominator), and the logistic regression decides when to stop extracting them (the decision). This separation of concerns keeps each model focused on what it does best.

---

7. What this approach adds

The extraction stopping calibration introduces two capabilities absent from the current graph RAG literature:

A formal probabilistic stopping criterion. Rather than heuristic gleaning loops (Microsoft GraphRAG), arbitrary pass limits, or fixed-cost budgets

(TERAG, E²GraphRAG), the system fits a calibrated logistic regression to historical yield data and uses a statistically grounded decision boundary. The threshold is not hand-tuned - it emerges from MLE on observed extraction patterns.

An explicit operational cost dimension. The adversarial position - extract everything, treat sections as atomic units - is the implicit assumption across GraphRAG, LightRAG, KGGen, and SocraticKG. The stopping calibration reframes extraction as a cost-benefit optimization: each additional LLM pass has a known token cost, and the sigmoid estimates the probability that the pass will yield new information. When that probability drops below 50%, the expected value of another pass is negative.

> **The design question this resolves:** "Is there a defensible point at which we can say 'there's no more juice to squeeze' and flag this section as completely extracted?" The answer is yes - and the defensibility comes from the calibrated model, not from a heuristic or an arbitrary pass count.

References

1. Giordano, L. & Razniewski, S. "Foundations of LLM Knowledge Materialization: Termination, Reproducibility, Robustness." arXiv:2510.06780, Findings of EACL 2026.
2. E²GraphRAG: Streamlining Graph-based RAG for High Efficiency and Effectiveness. arXiv:2505.24226, 2025.
3. TERAG: Token-Efficient Graph-Based Retrieval-Augmented Generation. arXiv:2509.18667, 2025.
4. Empowering GraphRAG with Knowledge Filtering and Integration. arXiv:2503.13804, 2025.
5. SocraticKG: Knowledge Graph Construction via QA-Driven Fact Extraction. arXiv:2601.10003, 2026.
6. Microsoft GraphRAG Methods. microsoft.github.io/graphrag.
7. LLM-empowered Knowledge Graph Construction: A Survey. arXiv:2510.20345, 2024.
8. Branzan, C. "From LLMs to Knowledge Graphs: Building Production-Ready Graph Systems in 2025." Medium, 2025.
9. "Beta-Binomial Model for Count Data: An Application in Estimating Model-Based Oral Reading Fluency." PMC, 2025.

10. Gebregziabher, M. et al. "A comparison of zero-inflated and hurdle models for modeling zero-inflated count data." J Stat Distrib App, 2021.
11. Lloyd-Smith, J.O. "Maximum Likelihood Estimation of the Negative Binomial Dispersion Parameter for Highly Overdispersed Data." PMC, 2007.
12. Robinson, M.D. & Smyth, G.K. "Small Sample Estimation of Negative Binomial Dispersion." Biostatistics, 2008.
13. Aggarwal, A. et al. "ConSol: Sequential Probability Ratio Testing to Find Consistent LLM Reasoning Paths Efficiently." arXiv:2503.17587, 2025.
14. Minka, T.P. "A comparison of numerical optimizers for logistic regression." Microsoft Research, 2003.
15. Pedregosa, F. "Numerical optimizers for Logistic Regression." Keep the gradient flowing, 2013.
16. Lin, C.-J. & Moré, J.J. "Newton's Method for Large Bound-Constrained Optimization Problems." JMLR 9, 2008.
17. Böhning, D. & Lindsay, B.G. "Monotonicity of quadratic-approximation algorithms." Ann. Inst. Stat. Math., 1988.