Autonomous Agents for Fair Second-Hand Price Prediction

Lucas Sorge
Student
Department of Computer Science
Iowa State University
lsorge13@iastate.edu

Tanish Visanagiri
Student
Department of Computer Science
Iowa State University
tanishv@iastate.edu

May 6, 2026

Abstract

Second-hand marketplace pricing is difficult because item value depends on incomplete, inconsistent, and highly contextual signals, including category, condition, brand, timing, location, market demand, visual defects, and seller behavior. A simple pricing model can return a number quickly, but speed alone does not make the estimate accurate, fair, or useful. This report studies whether a more agentic Throwly pricing pipeline can improve price prediction relative to Throwly's vanilla pricing model.

The case study uses Throwly's internal AI evaluation framework and reported experiment results to compare prompt engineering, architecture design, retrieval strategy, and model selection for second-hand price estimation. The benchmark dataset contains 2,831 eBay sold listings across 23 categories. The analysis focuses on three main measurements: mean absolute error (MAE), hit rate within an acceptable pricing band, and latency. These metrics capture the core product tradeoff: the system must be accurate enough to produce useful prices, consistent enough to place listings near realistic market values, and fast enough to support a smooth listing experience.

The results show that the vanilla model is not sufficient by itself. Stronger performance comes from combining structured prompts, item metadata, retrieval from sold-listing data, and careful model selection. Retrieval-grounded configurations improve the quality of price estimates by giving the model access to real marketplace context, while metadata-aware architecture improves performance without adding unnecessary complexity. Model choice also affects the tradeoff between pricing quality

and response time. Overall, the evidence supports the thesis that autonomous AI pricing agents can produce more accurate and consistent second-hand price estimates than a vanilla model, but only when the system is grounded in structured item information, market data, and deployment-aware evaluation.

1 Introduction

Pricing is one of the central problems in second-hand marketplaces. Unlike retail pricing, second-hand pricing does not usually have a stable manufacturer price, uniform condition, or complete product specification. Two listings may share the same broad category but differ in value because of damage, brand, model, age, accessories, local demand, seasonality, trend behavior, and timing. For a platform such as Throwly, which helps users list second-hand items, this creates a direct product problem: sellers need a price that feels fair and realistic, while buyers need listings that are not inflated beyond market value.

A bad price damages marketplace trust. If the suggested price is too low, the seller may feel that the app undervalues the item. If the suggested price is too high, buyers may ignore the listing and the marketplace may become less liquid. The pricing model therefore affects more than prediction accuracy. It influences user confidence, listing completion, search quality, and whether the marketplace feels credible.

Traditional non-agentic approaches can partially address pricing, but each has important limitations. A seller can choose a price manually, but this depends on individual knowledge and may be inconsistent. A marketplace can use fixed category ranges, but broad ranges ignore item-specific condition, brand, damage, and local demand. A system can use category averages, but averages are weak for categories with large price dispersion. A rule-based estimator can combine category and condition, but rules struggle with visual details and unusual items. These methods are simple and fast, but they do not adapt well to the heterogeneous signals that define second-hand value.

AI appears useful for this problem because modern language and vision-language systems can interpret images, item descriptions, condition metadata, and retrieved market evidence in the same workflow. However, a basic AI model placed behind an endpoint is not automatically a reliable pricing system. The provided Throwly results show that simple prompts, direct architectures, and unsupported model guesses produce weaker accuracy and less useful tradeoffs. The core engineering question is therefore not simply whether AI can generate prices. The question is whether an autonomous pricing agent can use structured prompts, metadata, retrieval, and model selection to outperform Throwly's vanilla baseline in a measurable way.

This report answers that question as a comprehensive case study. The comparison target is Throwly's own vanilla pricing setup.

1.1 Research Question

Can accurate and fair independent price prediction automation be achieved using autonomous agents, compared with Throwly's vanilla pricing model?

In this paper, *fair* does not mean demographic fairness. The available evidence does not support claims about protected classes or socioeconomic fairness. Instead, fair pricing means market-grounded, consistent, explainable, and less arbitrary than a simple unsupported model guess. A fair estimate should be close to comparable sold-listing evidence, should avoid obvious overpricing or underpricing, and should be produced with enough speed and reliability to support a real product workflow.

1.2 Thesis

Autonomous agents can improve Throwly's second-hand price prediction relative to a vanilla baseline when the pricing workflow combines structured prompts, item metadata, category-aware retrieval, sold-listing context, model comparison, and latency-aware deployment design. The evidence does not show that AI is universally better than humans; it shows that agentic, market-grounded AI performs better than simpler Throwly baseline configurations and provides a scalable path toward more consistent marketplace pricing.

2 Background: Why Second-Hand Pricing Is Hard

Second-hand pricing is hard because price is not only a property of an object. It is a property of an object in a market at a specific time and place. The same bike, laptop, desk, or jacket can have different reasonable prices depending on condition, completeness, location, urgency, brand recognition, and current demand. This makes the problem especially difficult for a college-focused marketplace, where users often list common but condition-sensitive items such as furniture, electronics, bikes, clothes, books, and appliances.

Several signals make pricing unstable:

- **Condition:** damage, wear, missing parts, stains, scratches, battery health, and age can change value sharply.
- **Category:** some categories have stable prices, while others have wide price ranges because of brand and specification differences.

- **Location:** a price that works in one local market may not work in another because demand and availability differ.
- **Timing:** move-out periods, school calendars, trends, and seasonality can change demand.
- **Information quality:** images may be unclear, titles may be missing, and descriptions may be short or incomplete.
- **User incentives:** sellers may overestimate sentimental value, while buyers may anchor on unusually low prices.

Because of these factors, a pricing system must do more than classify an image and output a number. It must infer the item, identify the category, interpret visual and textual signals, retrieve relevant comparables, filter noisy evidence, and return a price quickly enough that the user does not abandon the listing process.

3 Baseline and Non-Agentic Pricing Approaches

Before evaluating the autonomous agent design, it is necessary to define what the baseline is. In this paper, the baseline is Throwly's simpler or vanilla pricing setup: a production-style model path that relies on limited input, weak or no retrieval context, and less structured prompting. The retrieval-context experiment identifies this baseline as Prompt A, the production baseline using image and condition only. In that setting, Prompt A achieved weaker accuracy and hit-rate performance than retrieval-grounded alternatives.

Non-agentic pricing methods can be grouped into four broad categories.

3.1 Manual Seller Pricing

Manual seller pricing is the simplest approach. The seller decides what the item is worth. This requires no AI infrastructure, but it is inconsistent. Sellers may not know market value, may overprice based on purchase price, or may underprice when they need to sell quickly. Manual pricing can work for common items, but it does not scale into a consistent marketplace experience.

3.2 Fixed Category Ranges

A platform can assign price bands to broad categories. For example, a used chair, bike, or textbook could receive a suggested range. This approach is fast and easy to explain, but

it ignores important variation inside each category. A basic chair and a designer chair are not equivalent. A damaged bike and a well-maintained bike should not receive the same recommendation.

3.3 Category Averages

A system can calculate an average sold price by category. This is more data-driven than fixed ranges, but averages are often too broad. They are especially weak for categories such as computers, bicycles, cameras, and furniture, where brand, model, condition, and specifications create wide price dispersion.

3.4 Rule-Based Estimation

A rule-based system can adjust price based on category, condition, age, or other metadata. This is transparent, but it requires manual rule maintenance and struggles with visual condition cues. It also performs poorly for rare or unusual items because rules cannot cover every edge case.

These alternatives show why the problem is now appropriate for AI. Pricing requires both structured signals and flexible interpretation. However, the Throwly evidence also shows that basic AI is not enough. The strongest results come from agentic workflows that combine model inference with structured context, retrieval, and evaluation.

4 Throwly System Overview

Throwly is a second-hand marketplace application that uses AI to help users estimate listing prices. The pricing feature is designed to reduce seller uncertainty and make listing creation easier. The system receives an image and available item metadata, then returns a suggested price. In a stronger agentic setup, the workflow can also select a supported category, use location and timestamp context, generate a search query, retrieve marketplace comparables, clean and subsample data, and produce a grounded price estimate.

Figure 1 shows the Throwly architecture concept. The flow begins with an item, such as a bike, then performs data cleaning and uses image and optional text to identify the category. The category is checked against a supported category database. The system then uses category, location, and timestamp context to form a query. It can retrieve supplementary data from a search API such as eBay, clean that data, combine it with proprietary data, and pass a subsample to the model. The model produces a price and can optionally explain

the output for interpretability. User feedback can then be routed into a correction loop and future training or database improvement.

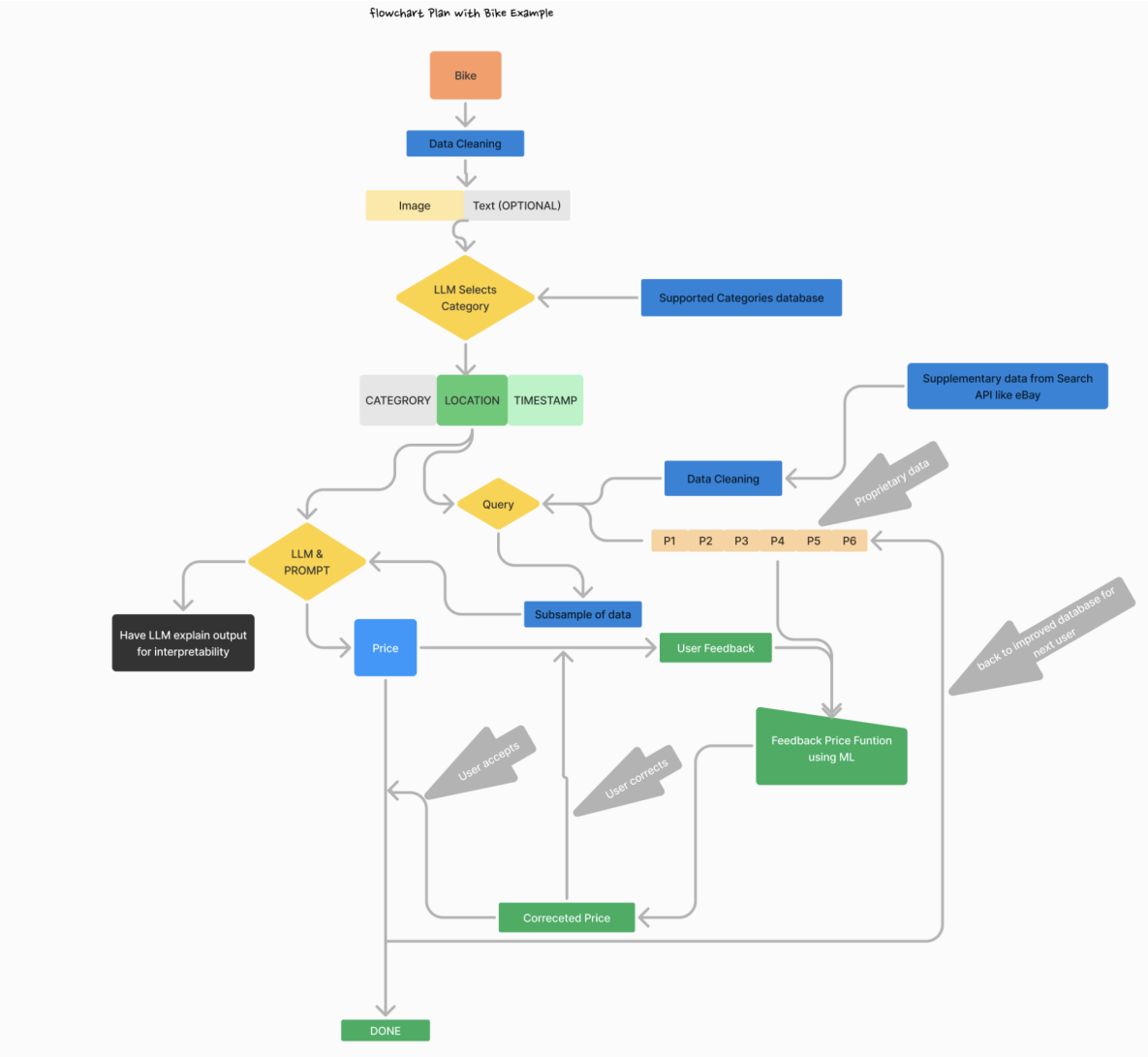


Figure 1: Throwly agentic pricing workflow concept. The diagram shows category selection, query generation, data cleaning, retrieval, pricing, explanation, and feedback loops.

This architecture matters because it separates a vanilla AI endpoint from a pricing agent. A vanilla endpoint receives input and predicts a price. An autonomous pricing agent coordinates intermediate steps: category selection, retrieval, filtering, prompt construction, model selection, logging, and feedback. In the second-hand pricing context, those intermediate steps are not optional engineering details. They are the mechanism that allows the model to use current and relevant market evidence.

5 Contributions

This case study identifies five main contributions from the Throwly evaluation work.

1. **Evaluation dataset:** Throwly created a benchmark dataset of 2,831 eBay sold listings across 23 categories. Sold listings serve as ground-truth market evidence because they reflect completed transactions rather than only asking prices.
2. **Evaluation framework:** The project evaluates pricing systems across accuracy, percentage error, acceptable-band hit rate, latency, and operational success rate.
3. **Prompt and architecture experimentation:** The framework tests multiple prompt families and architectural designs rather than assuming a single prompt or endpoint is sufficient.
4. **Retrieval grounding:** The experiments compare baseline prediction with retrieval-augmented configurations that use static sold-listing retrieval and live eBay retrieval.
5. **Deployment-oriented model comparison:** The model study compares accuracy, speed, hit rate, and completion behavior across Claude Opus 4.6, Claude Sonnet 4.6, and Claude Haiku 4.5.

The overall contribution is not a claim that AI pricing is solved. The contribution is a structured method for testing which combinations of prompt, architecture, retrieval, and model design make AI pricing more accurate and practical than a vanilla baseline.

6 Methodology

6.1 Evaluation Design

The provided Throwly report organizes the evidence into three complementary tracks:

1. **Prompt and architecture study:** evaluates prompt families and architecture families under a zero-retrieval or limited-context setup.
2. **Retrieval-context study:** evaluates whether category guidance, static embedding retrieval, and live eBay retrieval improve performance over the production baseline.
3. **Cross-model study:** evaluates different Claude model variants on a separate benchmark slice.

This structure is appropriate because pricing quality depends on more than one system layer. A weak prompt can harm a strong model. A strong model can still fail if it lacks market context. Retrieval can improve accuracy but increase latency. A fast model can be unattractive if completion reliability or price quality is poor. The evaluation therefore treats price prediction as a system problem rather than a single-model problem.

6.2 Dataset

The benchmark dataset contains 2,831 sold listings collected from eBay across 23 categories. Only fixed-price sold items were retained and the data was cleaned by removing duplicates, unknown conditions, and poor-quality titles. The dataset was split by category into an 80% retrieval pool and a 20% held-out test set. The retrieval pool supports embedding-based comparable retrieval, while the held-out test set functions as ground truth during evaluation.

Using sold listings is important because sold prices are closer to market-clearing prices than active asking prices. Active listings can be biased upward because unsold items may remain listed precisely because their prices are too high. This matters for Throwly because the system should estimate what an item can reasonably sell for, not just what someone asked for.

The eBay Browse API supports item search by keyword, category, image, and item lookup, making it a practical source for marketplace retrieval workflows (eBay Developers, *Browse API Overview*). The search method also supports searching by keyword, category, eBay product ID, GTIN, and related parameters (eBay Developers, *Search*). These capabilities fit the Throwly workflow because the model can extract an item query and use retrieval to gather comparable marketplace evidence.

6.3 Metrics

The evaluation uses several metrics because no single metric captures the full product requirement.

- **Mean Absolute Error (MAE):** average dollar difference between predicted price and ground-truth price. Lower is better. MAE is easy to interpret because it remains in dollars. Scikit-learn defines MAE as a non-negative regression loss where the best value is 0.0 (scikit-learn Developers, *mean_absolute_error*).
- **Mean Absolute Percentage Error (MAPE):** average percentage difference between predicted price and ground-truth price. Lower is better. MAPE helps compare errors

across price levels, but it can be unstable for very low-price items because a small dollar error can become a large percentage error. Scikit-learn documents MAPE as a relative regression loss and notes that percentage values are represented as relative values rather than 0–100 values inside the function (scikit-learn Developers, *mean_absolute_percentage_error*).

- **Hit rate:** percentage of predictions within an acceptable pricing band. Higher is better. This is useful for product evaluation because users may tolerate small deviations but not large pricing misses.
- **Latency:** average response time. Lower is better because listing creation must feel fast.
- **Success rate:** percentage of cases that produce a valid output. Higher is better because failed predictions damage the product workflow.

MAE is the primary accuracy metric because the product output is a dollar price. MAPE is supplementary because it shows relative error. Hit rate is especially important for marketplace usability because a price does not need to be perfect to be useful; it needs to fall within a reasonable decision band.

6.4 Prompt Families

The prompt-and-architecture study evaluates six prompt families:

- `p_structured`: category-aware guidance with clearer instruction format.
- `p_confidence`: prompt variant that incorporates uncertainty framing.
- `p_reasoning`: prompt variant that encourages explanation and reasoning.
- `p_fast_minimal`: compressed prompt intended to reduce complexity.
- `p_strict_json`: prompt emphasizing rigid output formatting.
- `p_minimal`: minimal prompt with very limited guidance.

This comparison tests whether prompt structure matters. If a minimal prompt performed as well as a structured prompt, the system could favor simplicity. The results show the opposite: structure is valuable.

6.5 Architecture Families

The architecture study evaluates four architecture families:

- `arch_metadata`: prediction augmented with structured item metadata.
- `arch_ensemble`: multiple predictions combined into a final estimate.
- `arch_two_stage`: first extracts or describes the item, then prices it.
- `arch_direct`: simpler single-step prediction path.

This comparison tests whether additional architecture complexity improves pricing. The answer is mixed. Metadata augmentation clearly helps, but ensembling increases latency substantially, and two-stage processing does not outperform metadata augmentation.

6.6 Retrieval Strategies

The retrieval-context study evaluates four configurations:

- **Prompt A – Production Baseline:** image and condition only.
- **Prompt B – Price Bands:** category guidance and price bands.
- **Prompt C – Embedding Retrieval:** extracted query plus semantically similar sold listings from a static eBay dataset.
- **Prompt D – Live eBay Retrieval:** extracted query plus live eBay comparables.

This is the most important comparison for the research question because it directly tests the vanilla baseline against more agentic, context-grounded designs.

6.7 Model Selection

The model study compares Claude Opus 4.6, Claude Sonnet 4.6, and Claude Haiku 4.5. Anthropic describes Claude as a family of large language models with different performance and speed profiles (Anthropic, *Models Overview*). The official documentation identifies Sonnet 4.6 as a balanced model and Haiku 4.5 as a fast model for high-volume, latency-sensitive use cases (Anthropic, *Documentation*). In this case study, however, deployment choice is based on Throwly's own benchmark results rather than provider descriptions.

7 Results

7.1 Experiment Reliability

The prompt-and-architecture experiment produced 72 configuration summaries and 360 case-level outcomes. Across all runs, 275 of 360 predictions completed successfully, giving an overall success rate of 76.39%. The status breakdown was 209 direct successes, 66 fallback-direct successes, 65 API errors, and 20 ensemble failures. This reliability profile matters because an AI pricing system cannot be judged only by average accuracy on completed cases. A system that fails frequently is not production-ready even if its successful outputs look strong.

7.2 Prompt-Level Results

Table 1 summarizes the prompt-level results. The strongest prompt family by mean MAE was `p_structured`, with mean MAE of $119.36. The second-best was `p_confidence`, with mean MAE of $128.95 and the lowest mean MAPE at 65.52%. The most reliable prompt was `p_reasoning`, with a success rate of 93.33%. The compressed and rigid-format prompts performed worse. In particular, `p_strict_json` had the lowest latency but poor pricing quality, while `p_minimal` had the worst overall MAE.

Prompt	Mean MAE ($)	Mean MAPE (%)	Latency (s)	Best MAE ($)
`p_structured`	119.36	71.46	11.17	88.00
`p_confidence`	128.95	65.52	12.18	79.60
`p_reasoning`	141.78	71.98	11.19	111.00
`p_fast_minimal`	164.39	79.43	11.87	89.67
`p_strict_json`	190.94	90.46	9.85	103.00
`p_minimal`	197.13	90.05	11.74	101.00

Table 1: Prompt-level aggregates across evaluated configurations. Lower MAE and MAPE are better; lower latency is better.

The main conclusion is that prompt design materially affects price quality. The structured prompt does not win because it is the fastest. It wins because it provides the best average dollar accuracy while remaining usable in latency terms. This supports the broader claim that AI pricing requires task-specific instruction design rather than a generic prompt.

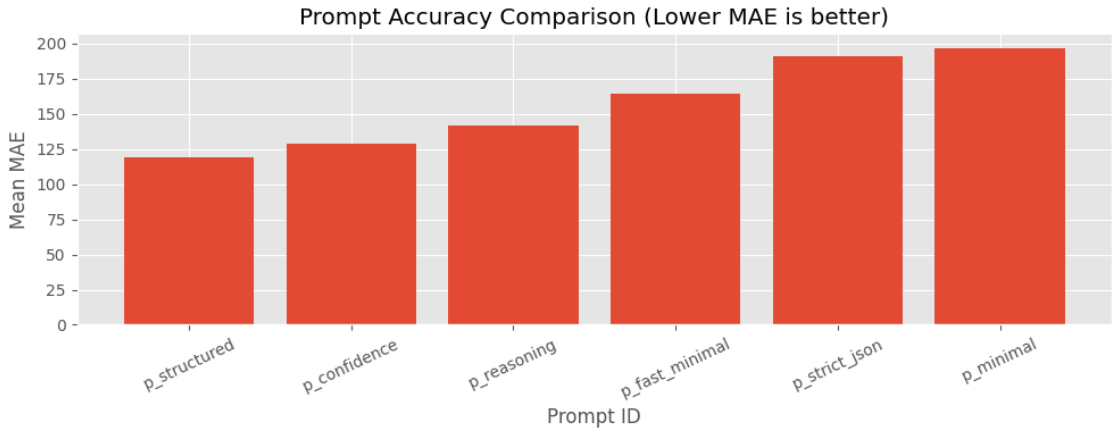


Figure 2: Prompt accuracy comparison. The structured prompt has the lowest mean MAE.

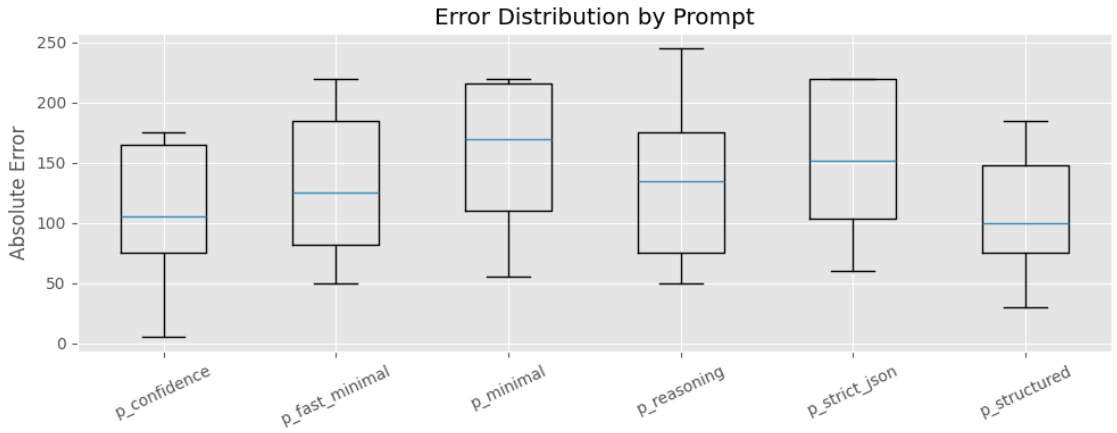


Figure 3: Absolute error distribution by prompt family. Structured and confidence prompts keep lower central error than minimal and strict JSON prompts.

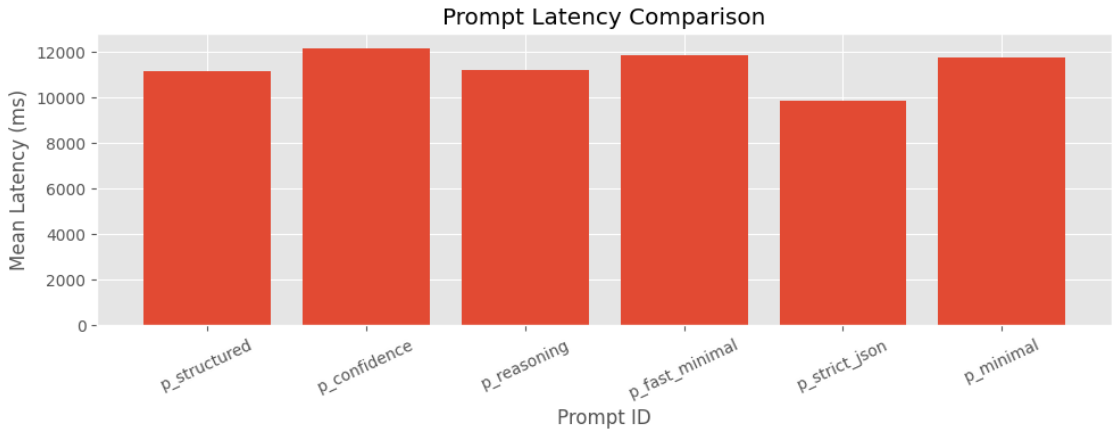


Figure 4: Prompt latency comparison. Lower latency alone does not identify the best prompt.

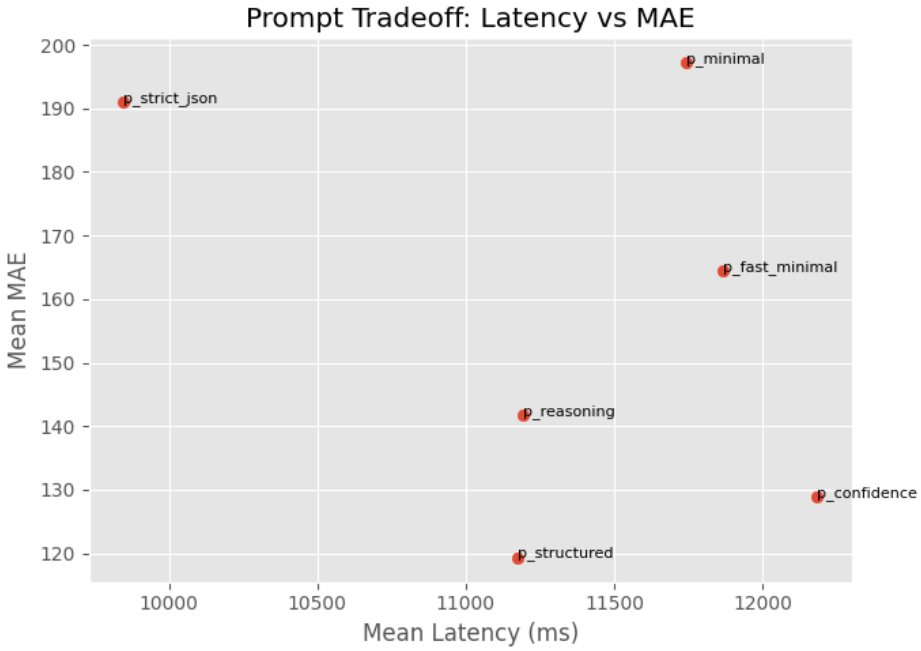


Figure 5: Prompt latency-versus-MAE tradeoff. The preferred region is lower-left: low error and low latency.

The prompt tradeoff plot shows that minimality is not enough. `p_strict_json` is faster than the other prompt families, but it has much worse error. `p_minimal` is both inaccurate and not meaningfully faster. `p_structured` is the best default candidate because it stays in the lower-error region without severe latency cost.

7.3 Architecture-Level Results

The architecture-level results are clearer than the prompt-level results. The best architecture family was `arch_metadata`, with mean MAE of approximately $130.9 and mean latency around 8.4 seconds. The next-best architecture by MAE was `arch_ensemble`, but its latency was much higher at roughly 18.8 seconds. `arch_two_stage` and `arch_direct` performed worse on MAE, and `arch_direct` was dominated by the metadata architecture on both accuracy and latency.

Architecture	Mean MAE ($)	Mean Latency (ms)
`arch_metadata`	130.9	8400
`arch_ensemble`	152.2	18800
`arch_two_stage`	160.9	8500
`arch_direct`	172.2	9800

Table 2: Architecture-level aggregates. Metadata augmentation is the strongest non-retrieval design.

The architecture results show that useful structure beats unsupported complexity. Metadata augmentation improves the input signal in a way that directly helps pricing. Ensembling can improve over weaker designs, but its runtime cost is too high for a default product path unless accuracy is prioritized far above speed. Two-stage reasoning also fails to justify itself in the reported results. The best non-retrieval architecture is therefore metadata augmented, not ensemble or direct.

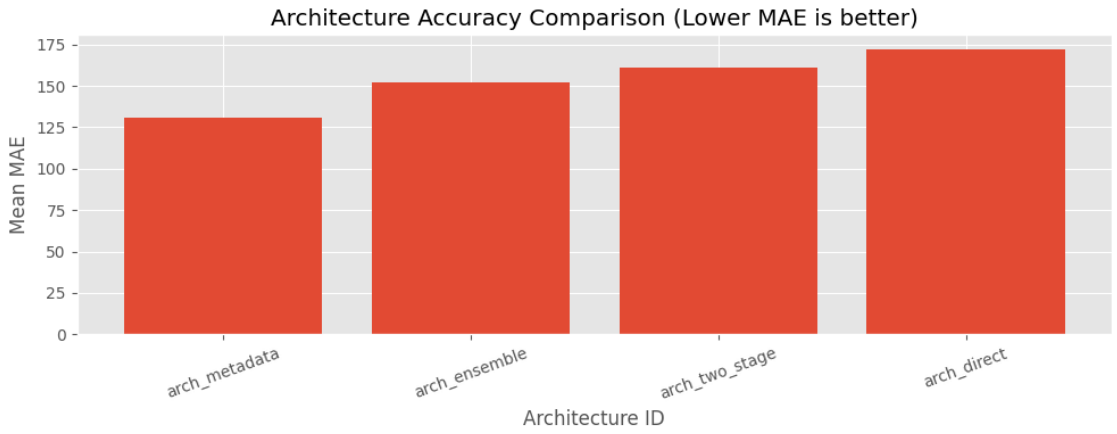


Figure 6: Architecture accuracy comparison. Metadata augmentation has the lowest mean MAE.

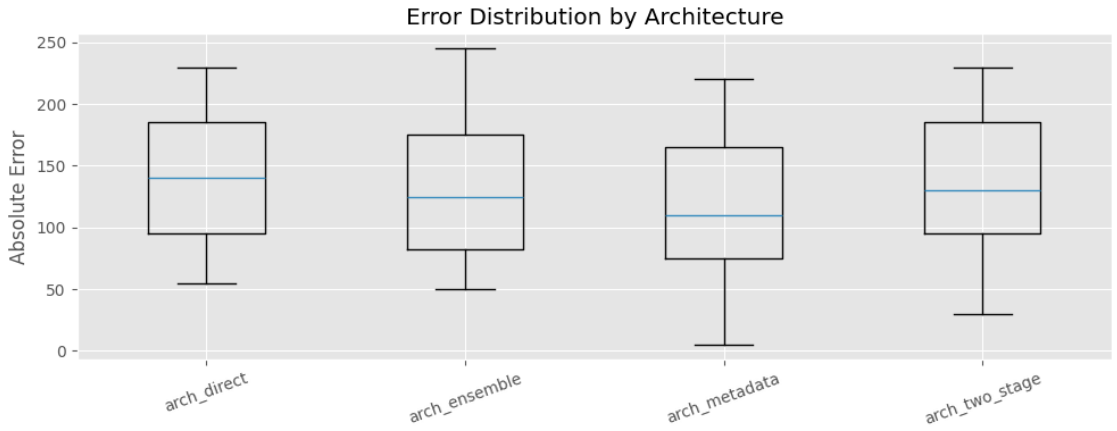


Figure 7: Absolute error distribution by architecture. Metadata has the most favorable central error behavior.

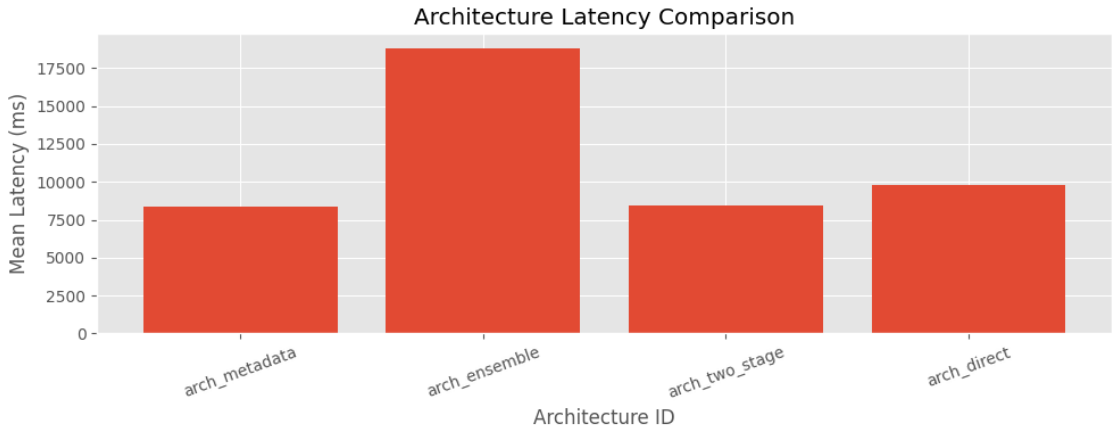


Figure 8: Architecture latency comparison. Ensemble is substantially slower than the other architecture families.

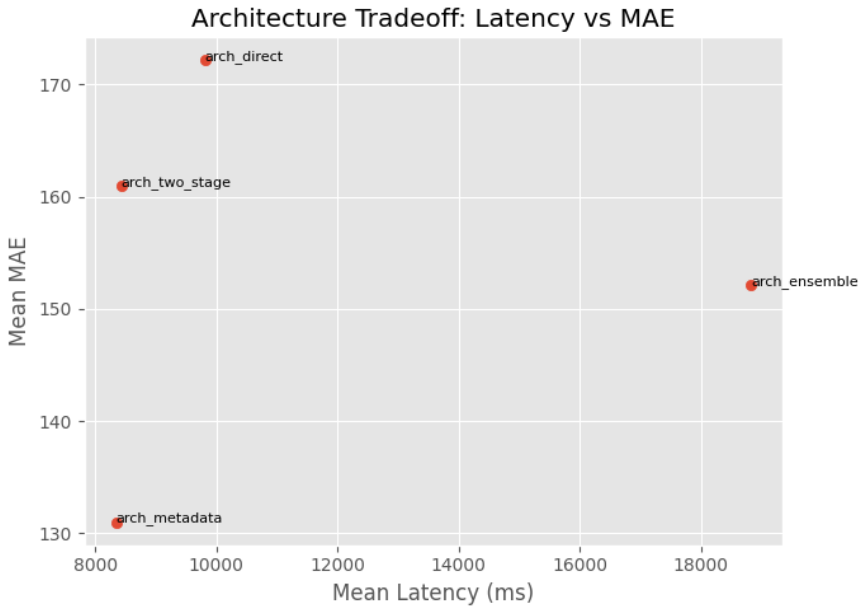


Figure 9: Architecture latency-versus-MAE tradeoff. Metadata occupies the most favorable region.

The architecture tradeoff plot supports a strong design conclusion: if Throwly must choose a non-retrieval architecture, `arch_metadata` is the best default. It improves over the direct baseline and avoids the high latency of ensemble prediction.

7.4 Retrieval-Context Results

The retrieval-context study directly compares the production baseline with richer context strategies. Table 3 reports the consolidated results from the provided draft.

Prompt Context	MAE ($)	MAPE (%)	Hit Rate (%)	Latency (s)
A – Production Baseline	112	65.1	19	9.43
B – Price Bands	103	76.7	22	9.24
C – Embedding Retrieval	89	72.8	32	10.75
D – Live eBay Retrieval	96	78.3	34	9.85

Table 3: Retrieval-context comparison. Retrieval-grounded prompts outperform the production baseline on MAE or hit rate. Lower MAE and latency are better; higher hit rate is better.

These results show that retrieval improves the vanilla system. Prompt A, the production baseline, had the weakest hit rate and worse MAE than all three alternatives. Prompt C, embedding retrieval over sold listings, achieved the best raw MAE at $89. Prompt D, live

eBay retrieval, achieved the best hit rate at 34%. This is important because a product may value hit rate as much as raw MAE: a price inside an acceptable band is often useful even when it is not exactly correct.

The retrieval results also show why active listing data must be handled carefully. Live eBay retrieval can improve hit rate by giving the model fresh context, but active listings may be biased upward if asking prices exceed final sale prices. Static sold-listing retrieval may be better for raw accuracy because sold listings are closer to realized transaction prices. For Throwly, the likely best long-term path is not choosing one source permanently. It is combining clean sold-listing data, category-aware filters, recency signals, and live marketplace checks.

7.5 Model-Level Results

The model comparison adds another layer to the system design. Table 4 summarizes the reported model-level metrics. The figure set reports rounded visual values; the table uses the detailed values provided in the internal report where available and uses the updated visual values for the Haiku row where the attached model figure differs from the earlier draft.

Model	MAE ($)	MAPE (%)	Hit Rate $\leq 20\%$ (%)	Avg Latency (s)
Claude Opus 4.6	7.36	38.9	25.0	11.1
Claude Sonnet 4.6	7.75	39.3	33.1	7.3
Claude Haiku 4.5	18.5	92.7	4.6	6.1

Table 4: Cross-model comparison. Sonnet provides the best overall deployment tradeoff, while Opus provides the lowest MAE.

Claude Opus 4.6 has the lowest MAE and lowest MAPE, making it the best raw-accuracy model in the model slice. However, its latency is the highest. Claude Sonnet 4.6 is only slightly worse on MAE and MAPE, but it has the highest hit rate and much lower latency. This makes Sonnet the strongest operational default for Throwly under the current evidence. Claude Haiku 4.5 is faster but much less accurate and has the weakest hit rate, so it is not the best candidate for price quality despite its speed.

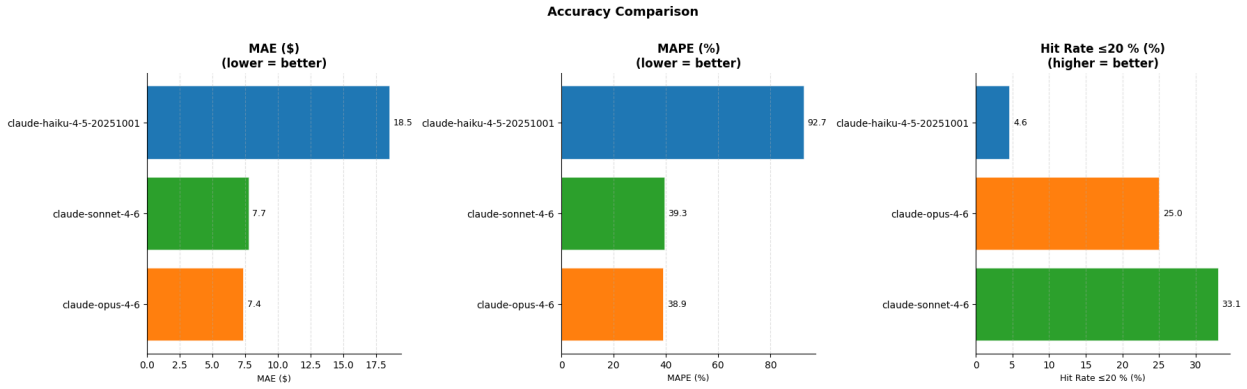


Figure 10: Model accuracy comparison across MAE, MAPE, and hit rate. Sonnet has the highest hit rate, while Opus has the lowest MAE.

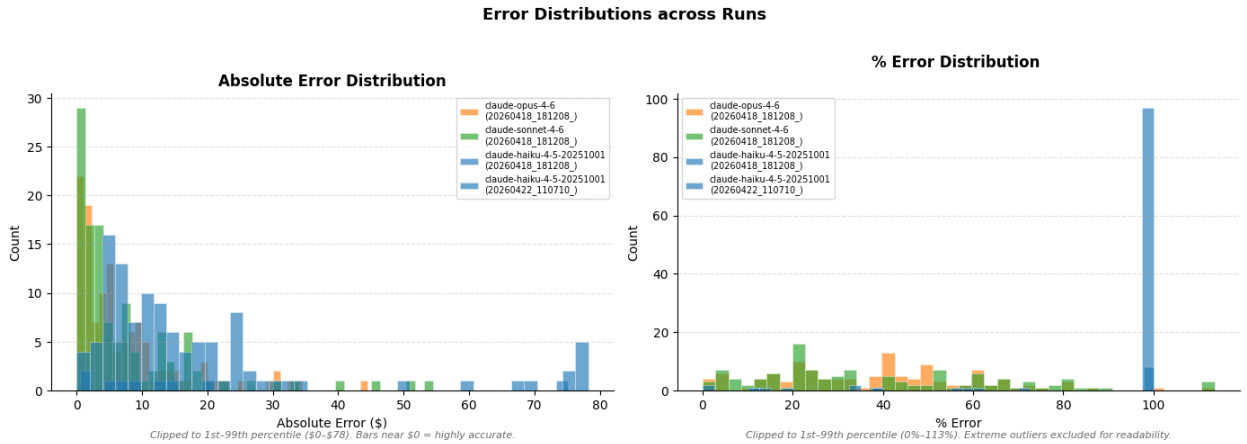


Figure 11: Error distributions across model runs. The distribution view shows both absolute and percentage error behavior.

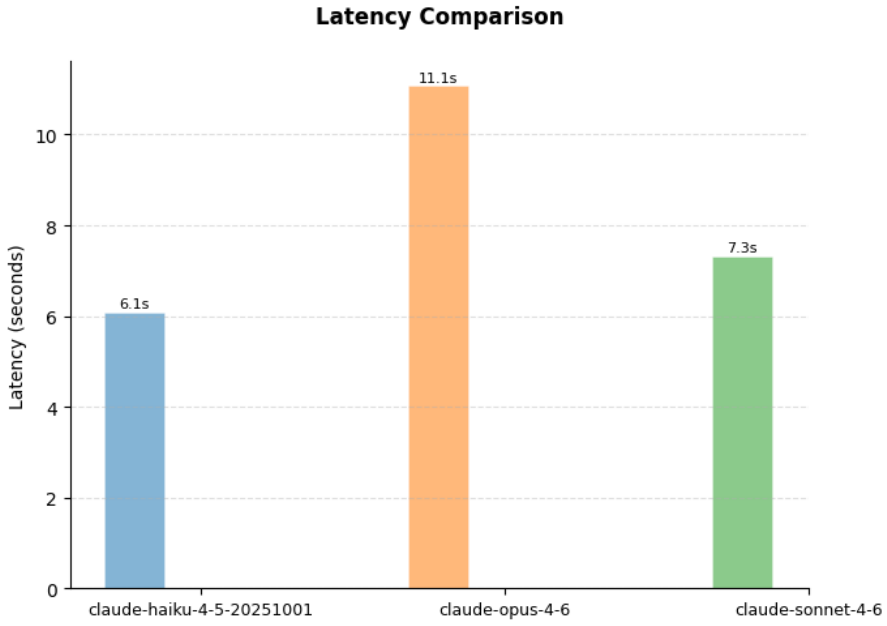


Figure 12: Average latency comparison across evaluated models. Haiku is fastest, but speed alone does not make it the best pricing model.

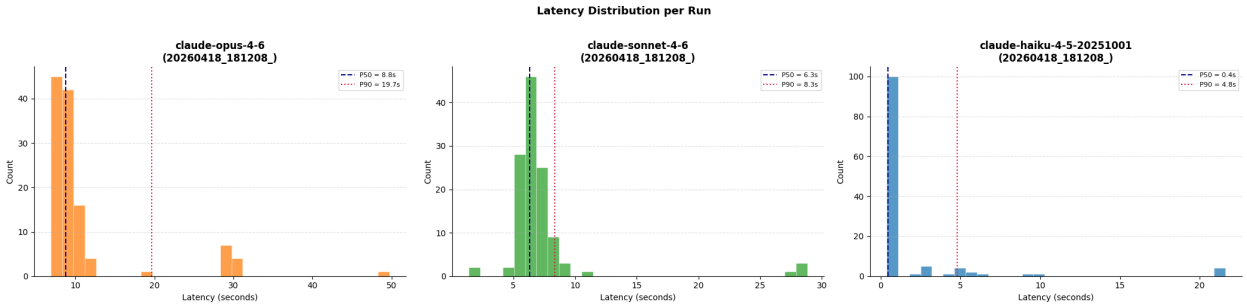


Figure 13: Latency distribution per model run, including p50 and p90 markers. Opus has the widest latency tail.

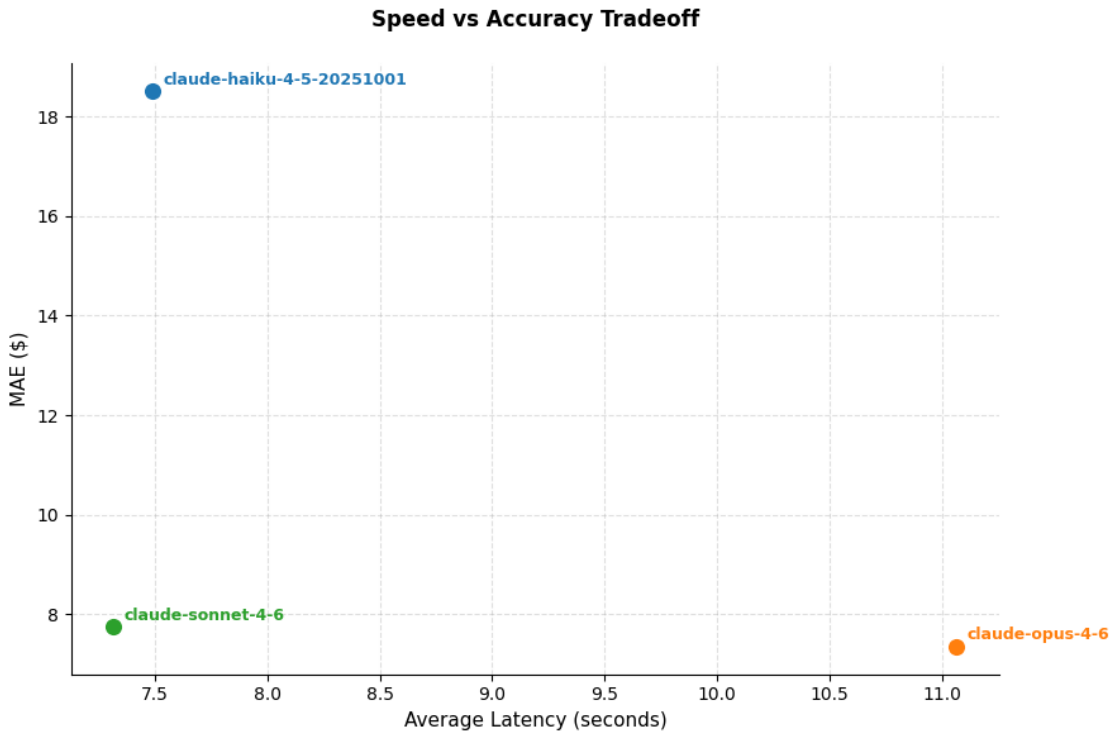


Figure 14: Speed-versus-accuracy tradeoff by model. Sonnet is closest to the preferred deployment balance.

The model comparison supports an important deployment conclusion: the best model is not necessarily the model with the lowest MAE. In a production marketplace, latency and hit rate matter. Sonnet is attractive because it remains close to Opus on error while improving response time and acceptable-band accuracy.

7.6 Category-Level Results

The category-level figures show that model performance is not uniform across the marketplace. Some categories are easier because items are low-variance or visually simple. Others are harder because value depends on brand, specification, condition, or rare details that may not be visible in a single image.

The most difficult categories include Bicycles, Computers, Furniture, Camera & Photo, Music & Accessories, and Cell Phones & Accessories. These categories have larger price ranges and stronger dependence on item-specific details. For example, a computer's value depends heavily on specifications, age, storage, battery, brand, and condition. A bike's value depends on frame type, components, brand, size, condition, and local demand. A single image may not capture all of these factors.

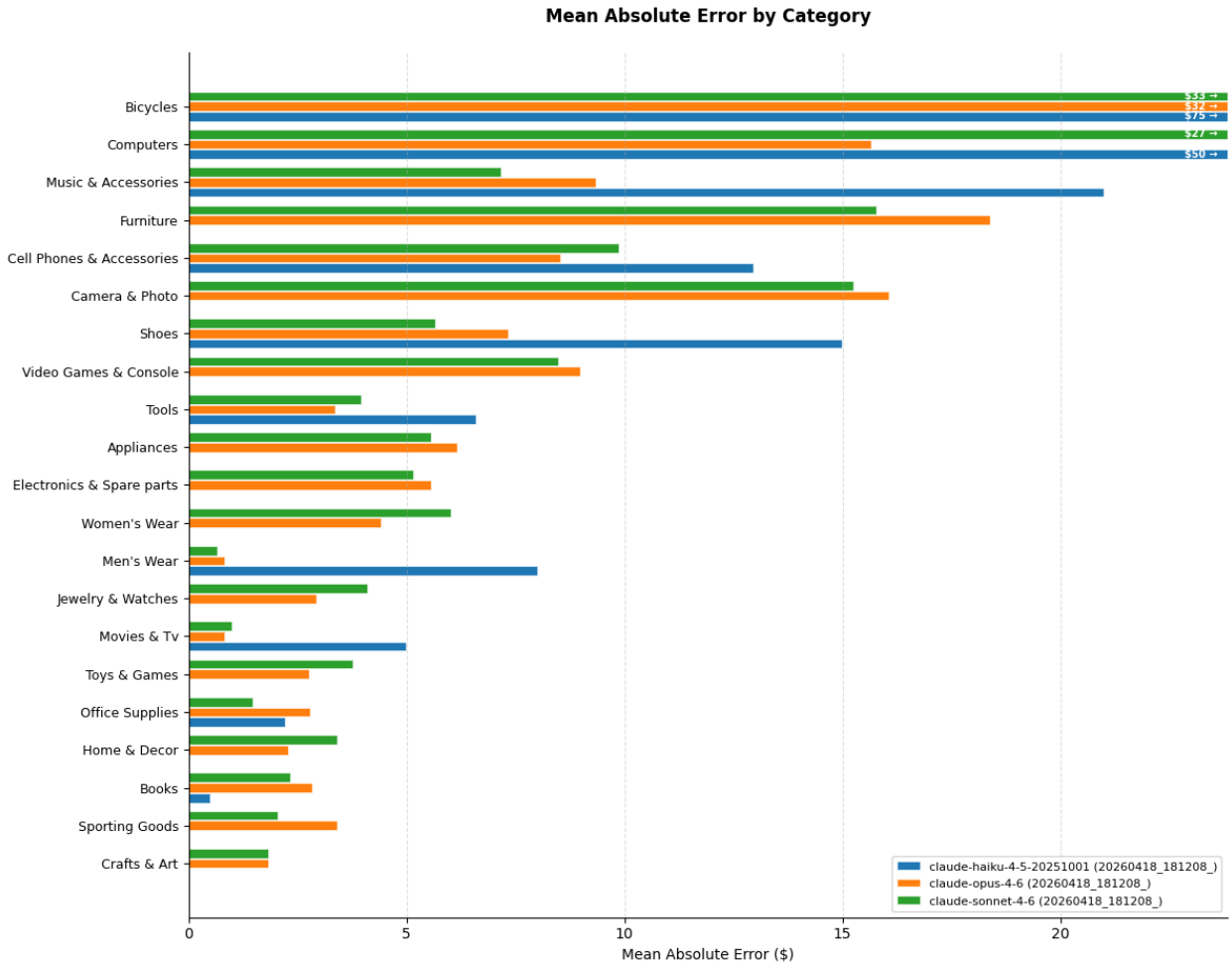


Figure 15: Mean absolute error by category. Performance varies substantially across categories.

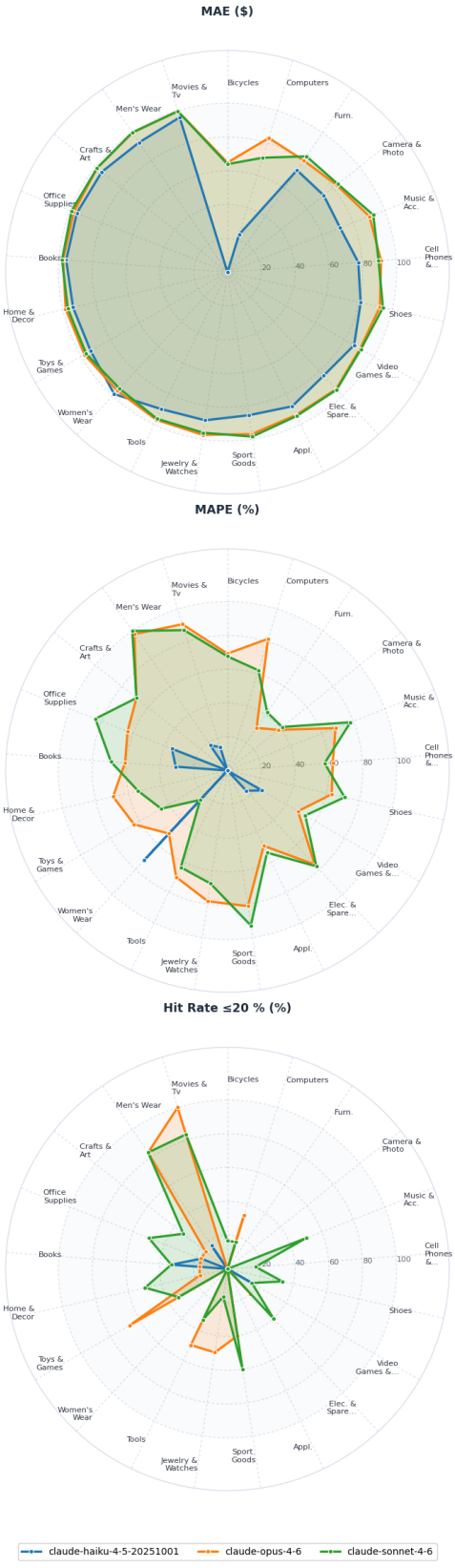


Figure 16: Category-level radar comparison across MAE, MAPE, and hit rate. No model dominates every category and metric.

The category-level evidence supports the need for niche datasets. A general pricing agent may perform adequately on common items, but expensive or specification-heavy categories need specialized retrieval, richer metadata, and stronger validation. This suggests a practical future direction: Throwly should improve per-category pipelines rather than assuming one global model configuration is optimal for every product type.

8 Discussion

8.1 Why the Baseline Struggles

The vanilla baseline struggles because it asks the model to infer market price from insufficient context. Image and condition are useful, but they do not fully determine value. The model may identify an item correctly but still lack information about comparable sales, local demand, age, brand, or market range. In categories with wide price dispersion, this missing context produces large errors.

The baseline also lacks strong guardrails. Without structured prompts, the model may rely too much on prior assumptions. Without retrieval, it may guess from general knowledge instead of current market evidence. Without metadata augmentation, it may underuse the signals that are available. Without logging and evaluation, it is difficult to know whether changes actually improve the system.

The results show that simply putting an AI model behind a pricing endpoint is not the solution. The solution is a measured pipeline where the model is only one component.

8.2 Why Structured Prompting Helps

Structured prompting helps because it reduces ambiguity. A structured prompt can tell the model what signals matter, how to treat condition, how to avoid unsupported certainty, and what output format to produce. The prompt-level results show that structure improves MAE compared with minimal and rigid-format prompts.

This does not mean longer prompts are always better. The results show that prompts must be optimized for the task. A prompt can be too minimal, too rigid, or too slow. The best prompt is the one that gives enough structure to improve pricing without creating unnecessary latency or failures. In the current evidence, `p_structured` is the strongest default.

8.3 Why Metadata Matters

Metadata matters because second-hand price is not visible from an image alone. Condition, category, model, rarity, brand, location, and timestamp can change the correct estimate. The metadata architecture performs best because it gives the model structured signals that are directly relevant to price. This is more effective than relying on a direct image-to-price path.

The architecture results also show that complexity must be useful. Ensemble prediction adds runtime, but the improvement does not justify the latency cost as a default deployment path. Two-stage processing adds structure, but it does not outperform metadata augmentation. The best architecture is therefore the one that adds relevant information, not simply more steps.

8.4 Why Retrieval Is Central

Retrieval is central because price prediction is a market-grounding problem. A model can recognize an object, but it still needs comparable evidence to estimate what that object should sell for. The retrieval-context study shows that embedding retrieval and live eBay retrieval both improve over the production baseline.

Prompt C performs best on raw MAE because it retrieves comparable sold listings. Prompt D performs best on hit rate because live retrieval adds fresh marketplace context. These results suggest that Throwly should treat retrieval as a core part of the agent, not an optional feature. The system should retrieve evidence, clean it, remove irrelevant items, and pass a focused sample to the model.

8.5 Why Model Choice Still Matters

Even with good prompts and retrieval, model choice matters. The model controls visual understanding, reasoning, extraction quality, and price synthesis. Opus has the best raw error, Sonnet has the strongest practical tradeoff, and Haiku is faster but weaker in price quality. This creates an operational frontier rather than a single obvious answer.

For Throwly, the best default model should balance accuracy, hit rate, latency, and reliability. Based on the provided evidence, Claude Sonnet 4.6 is the strongest default. Opus may be useful for high-value or difficult categories where accuracy matters more than speed. Haiku may be useful only if future prompt and retrieval improvements can recover accuracy.

8.6 Fairness and Independence

In this paper, fairness means consistency and market grounding. A fair price estimate should not depend only on a seller's guess or a model's unsupported prior. It should use comparable sold listings, item metadata, and category-specific evidence. This can make the process more independent from arbitrary individual judgment.

However, fairness claims must remain limited. The provided evidence does not measure demographic bias, buyer outcomes, seller satisfaction, or human-versus-agent fairness. It only supports a narrower claim: Throwly's agentic configurations can produce more accurate and more consistent prices than the vanilla baseline under the reported metrics.

9 Use Cases

9.1 Listing Creation for Common Items

The most direct use case is helping sellers price items during listing creation. A student listing a chair, lamp, desk, textbook, appliance, or jacket may not know the correct price. The agent can use category, condition, and comparable listings to suggest a reasonable starting price. This reduces friction and helps listings go live faster.

9.2 Buying and Selling on Local Marketplaces

A user buying a car part, bike, phone, or furniture item on Facebook Marketplace or another local platform can use the agent as a price-checking tool. The system can compare the item against market evidence and warn the user if the listing appears high or low relative to comparable sales. This use case requires strong category and location awareness.

9.3 Unfamiliar Items

Users often need to price objects they do not know well, such as a light fixture, tool, instrument accessory, collectible, or specialized equipment. A pricing agent can identify the likely category and retrieve comparable items. This is where visual recognition and retrieval are especially useful.

9.4 Marketplace Trust and Consistency

At the platform level, pricing automation can make listings more consistent. If suggested prices are grounded in comparable data, the marketplace may have fewer extreme overprices

and fewer accidental underprices. This can improve buyer trust and seller confidence.

10 Deployment and Architecture Considerations

10.1 Speed

Latency is a product constraint. A pricing suggestion that takes too long can interrupt listing creation. The experiments show that latency varies substantially across architectures and models. Ensemble architectures and Opus-style models may improve quality, but they are slower. A production system may need to use a tiered strategy: fast default model for common items, stronger model for difficult or high-value categories, and fallback behavior for uncertain predictions.

10.2 Scale

Scaling the system requires controlling context size, retrieval cost, and API calls. Adding more data is useful only if the agent can retrieve the right subset. The Throwly workflow should avoid passing too many comparables directly to the model. Instead, it should query the correct supported category, clean retrieved listings, remove outliers, and pass a concise subsample.

10.3 Security and Access Control

A pricing API should include access control, rate limits, logging, and abuse prevention.

10.4 Explainability and Logging

Explainability is useful because users may want to know why a price was suggested. The workflow includes an option for the model to explain the output for interpretability. In production, the explanation should be short and evidence-based: category, condition, comparable prices, and uncertainty. Logging is also necessary so developers can track failures, latency, retrieved evidence, and model outputs.

11 Challenges and Limitations

11.1 No Human Baseline

This paper does not claim that Throwly's agent is better than human sellers or human appraisers. The provided evidence does not include a controlled human benchmark. The correct comparison is against Throwly's production baseline, direct architecture, minimal prompts, and other non-agentic or weaker configurations.

11.2 Evaluation Versus Production

An evaluation system can differ from a production system. Evaluation may use cleaned data, controlled categories, and selected benchmark cases. Production must handle blurry images, missing metadata, unusual items, API failures, adversarial inputs, and user impatience. The benchmark results are useful, but they do not eliminate the need for production monitoring.

11.3 Category Difficulty

Some categories are much harder than others. Expensive and specification-heavy categories produce larger errors. This limits the usefulness of one global model score. A model that performs well on low-variance categories may still fail on computers, bikes, cameras, or furniture. Future evaluation should report category-specific performance consistently.

11.4 Price Band Bias

The finds indicate that price-band data can make the system's suggested price lower than market value. This is plausible because broad bands can compress estimates toward safe or conservative ranges. However, the current evidence does not include a detailed ablation proving this mechanism. It should be treated as an observed concern that needs further testing.

11.5 Visual Identification Errors

The system can fail if the object is not centered, the lighting is poor, the image contains multiple objects, or the model identifies the wrong category. These issues are especially important for a camera-based listing flow. The agent should include confidence checks and ask for more information when the image is ambiguous.

11.6 Retrieval Noise

Retrieval can improve performance, but poor retrieval can harm it. If the search query retrieves wrong categories, active listings instead of sold listings, duplicates, outliers, or irrelevant brands, the model may anchor on bad evidence. Data cleaning and category filtering are therefore essential parts of the agentic pipeline.

11.7 Metric Limitations

MAE is interpretable but can overweight high-price categories. MAPE is useful for relative error but unstable for low-price items. Hit rate is product-friendly but depends on the chosen acceptable band. Latency averages can hide long-tail delays. For that reason, Throwly should continue reporting multiple metrics rather than optimizing only one.

12 Future Work

Future work should focus on strengthening the agent rather than only swapping models.

1. **Improve hit rate:** Optimize for acceptable-band accuracy because that maps closely to product usefulness.
2. **Expand sold-listing data:** Add more sold listings, especially for difficult categories such as bikes, computers, cameras, and furniture.
3. **Improve category-specific pipelines:** Use specialized extraction fields for categories where price depends on specifications, such as computer RAM, storage, bicycle components, or camera lens details.
4. **Test cheaper models carefully:** Cheaper models are attractive only if they preserve hit rate, completion reliability, and calibration.
5. **Add retrieval ablations:** Compare sold listings, active listings, internal marketplace data, and mixed retrieval under the same benchmark.
6. **Strengthen production monitoring:** Track latency, API errors, fallback behavior, user corrections, accepted prices, and final sale outcomes.
7. **Improve image robustness:** Test poor lighting, multiple objects, occlusion, missing text, and off-center items.

8. **Develop feedback training data:** Use accepted and corrected prices carefully, separating user preference from true market-clearing price.

The current research question should remain focused on the measurable baseline: whether Throwly's autonomous agent improves over Throwly's vanilla production path.

13 Conclusion

The evidence supports a clear answer to the revised research question: autonomous agents can improve second-hand price prediction relative to Throwly's vanilla baseline, but only when the system is designed as a grounded pricing workflow rather than a simple AI endpoint. The baseline struggles because image and condition alone do not provide enough market context. The stronger configurations improve performance by adding structured prompts, metadata, retrieval, and model selection.

The prompt study shows that `p_structured` is the strongest average prompt family, with mean MAE near $119. The architecture study shows that `arch_metadata` is the best non-retrieval architecture, with mean MAE near $130.9 and latency near 8.4 seconds. The retrieval-context study shows that retrieval-grounded prompts outperform the production baseline, with embedding retrieval achieving the best raw MAE and live eBay retrieval achieving the best hit rate. The model study shows that Claude Sonnet 4.6 is the strongest deployment default because it balances accuracy, hit rate, and latency better than the alternatives.

The main conclusion is therefore not that AI automatically prices better. The conclusion is that AI pricing becomes useful when it is agentic and evidence-grounded. A reliable system must identify the item, use metadata, retrieve relevant comparable sales, clean noisy data, select an appropriate model, measure tradeoffs, and log failures. Under that design, Throwly can move beyond vanilla pricing toward fairer, more consistent, and more scalable second-hand price automation.

Works Cited

Anthropic. "Documentation." *Claude API Docs*, Anthropic, https://platform.claude.com/docs/en/home. Accessed 5 May 2026.

Anthropic. "Models Overview." *Claude API Docs*, Anthropic, https://platform.claude.com/docs/en/about-claude/models/overview. Accessed 5 May 2026.

eBay Developers. "Browse API Overview." *eBay Developers Program*, eBay, https://developer.ebay.com/api-docs/buy/browse/overview.html. Accessed 5 May 2026.

eBay Developers. "Search: Browse API." *eBay Developers Program*, eBay, https://developer.ebay.com/api-docs/buy/browse/resources/item_summary/methods/search. Accessed 5 May 2026.

scikit-learn Developers. "mean_absolute_error." *scikit-learn Documentation*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html. Accessed 5 May 2026.

scikit-learn Developers. "mean_absolute_percentage_error." *scikit-learn Documentation*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_percentage_error.html. Accessed 5 May 2026.