

KotaAgent: An agent submitted to the ANAC 2026 SCM league

Hibino Kota¹, Takanobu Otsuka²

^{1,2}Nagoya Institute of Technology, Aichi, Japan

¹hibino.kota@otsukalab.nitech.ac.jp

²otsuka.takanobu@nitech.ac.jp

June 24, 2026

Abstract

This paper presents KotaAgent, an autonomous agent for the ANAC 2026 SCML Standard track. The agent enforces profit margins, uses delivery window constraints to reduce inventory risk, and applies fallback purchasing to avoid production failures when profitable trades are impossible. Buy and sell price thresholds are independently adapted using exponential moving average (EMA) and relaxed after negotiation failures. Across five independent tournament runs, KotaAgent outperforms the stronger **GreedyStdAgent** baseline in every run, and a dedicated multi-run study shows that fallback purchasing raises the worst-case score without reducing the mean.

1 Introduction

In the ANAC SCML Standard track, agents manage factories by negotiating contracts with suppliers and consumers and scheduling production over a fixed horizon. The main challenges are inventory costs and penalties for failing to satisfy sales contracts.

A difficult case occurs when supplier prices exceed consumer prices, making profitable trades impossible. KotaAgent addresses this using profit margin constraints, fallback purchasing, delivery window control, and adaptive price thresholds.

2 The Design of KotaAgent

KotaAgent extends `StdSyncAgent` and negotiates with all partners simultaneously using `first_proposals()` and `counter_all()`.

2.1 Negotiation Strategy

Profit margin enforcement. The agent calculates acceptable buy and sell price ranges. Purchases are limited by a price ceiling, while sales require a price floor that covers input costs. Offers outside these ranges are rejected.

Adaptive price fraction. KotaAgent maintains independent buy and sell fractions, initialized to 0.5 and limited to [0.1,0.9]. These fractions determine the initial proposal position within acceptable ranges. During negotiation, acceptance thresholds gradually move toward the boundary as the deadline approaches.

EMA learning. After successful negotiations, the corresponding fraction is updated toward the agreed price position using EMA with weights 0.7 and 0.3. After failures, the fraction is increased by 0.05 to relax future proposals.

2.2 Concurrent Negotiation

The agent distributes required quantities among partners and tracks accepted quantities during `counter_all()`. Once sufficient contracts are obtained, additional offers are rejected to prevent over-contracting. Future deliveries are handled separately.

Because all partners are negotiated with synchronously, multiple offers within the same step can be accepted simultaneously and lead to over-contracting. KotaAgent mitigates this by decrementing the remaining required quantity as each acceptance is recorded inside `counter_all()`, so the within-step needs are updated before the remaining offers are evaluated. This is a needs-based, implicit treatment of within-step simultaneity; an explicit step-level inventory account or a per-step reservation variable would handle the same problem more directly and is left as future work.

2.3 Risk Management

Delivery window constraint. Offers with delivery dates more than four steps ahead are rejected to reduce inventory risk.

Fallback buying. When profitable purchasing is impossible, KotaAgent accepts minimum supplier prices instead of stopping production. This avoids large shortfall penalties caused by zero production.

3 Evaluation

KotaAgent was evaluated with the `anac2024_std` tournament runner (n steps = 50, n configs = 5, n runs per world = 1) against three baselines: **GreedyStdAgent** (the stronger built-in baseline used by the rest of the cohort), **SyncRandomStdAgent**, and **RandomOneShotAgent**. To account for run-to-run variance, the tournament was repeated for five independent runs and we report the average, the standard deviation (SD), the minimum (worst-case), and the maximum (best-case) score over the five runs. Scores are normalized so that 1.0 corresponds to a break-even factory (revenue equal to cost); values above 1.0 indicate profit and negative values indicate net loss.

3.1 Main Results

Agent	Average	SD	Min	Max
KotaAgent	0.910	0.080	0.853	1.051
GreedyStdAgent	0.747	0.096	0.589	0.829
SyncRandomStdAgent	0.671	0.116	0.478	0.761
RandomOneShotAgent	-0.169	0.298	-0.473	0.233

Table 1: Normalized scores over five independent runs (n steps = 50, n configs = 5). SD is the run-to-run standard deviation.

KotaAgent achieved the highest score in every run and beat **GreedyStdAgent** in all five runs (margins +0.037 to +0.313). This confirms that the agent is competitive against a stronger, non-random opponent and not only against a random baseline.

3.2 Component Ablation

To separate the contribution of each component, the agent was built up one feature at a time. All rows are five-run averages under the unified setup of Table 1, so the increments are directly comparable and consistent with Tables 1 and 3.

Configuration	KotaAgent
Base	0.879
+ Delivery window	0.860
+ Separate fractions + failure learning	0.906
+ Fallback buying	0.910

Table 2: Incremental ablation. All rows are five-run averages under the unified setup of Table 1; the last two rows match the averages in Table 3.

Separate buy/sell adaptation gives the clearest gain. Under the unified five-run setup the full configuration with fallback (0.910) is the best overall and slightly exceeds the configuration without fallback (0.906); the two differ by only 0.004, well within the run-to-run standard deviation of 0.080, so their averages are effectively tied. We examine the fallback component in detail, including its effect on the worst-case score, in the next section.

3.3 Effect of Fallback Buying

To evaluate fallback buying properly, the full agent (with fallback) and an otherwise identical agent without fallback were each run five times under the unified setup of Table 1.

Configuration	Average	SD	Min	Max
With fallback (final)	0.910	0.080	0.853	1.051
Without fallback	0.906	0.077	0.812	0.998

Table 3: Fallback vs. no-fallback over five runs each.

The mean difference is only +0.004, far below the combined standard error (≈ 0.05), so the two configurations are statistically indistinguishable in mean score. Fallback buying therefore does *not* reduce average profit. At the same time, the worst-case (minimum) score is higher with fallback (0.853 vs. 0.812), while the standard deviation is essentially unchanged (0.080 vs. 0.077). Fallback buying thus acts as a low-cost insurance that raises the downside without sacrificing the mean; we do not claim a reduction in variance. A stronger test would report the per-world minimum score (rather than the per-run minimum), which would isolate the zero-production worlds that fallback is designed to rescue; this is left as future work.

3.4 Effect of the Delivery Window

In the ablation the four-step delivery window slightly lowered the average score ($0.879 \rightarrow 0.860$, a difference within the run-to-run standard deviation). A likely explanation is that a four-step horizon is too strict: it rejects some far-dated but still profitable offers, so the inventory saving is outweighed by the lost trading opportunities. Tuning the window length, or making it adaptive to inventory level, is a natural next step.

4 Lessons and Suggestions

Robustness matters. Strict profitability constraints can cause zero production, while fallback buying raises the worst-case score at no measurable cost to the mean.

Test against stronger, non-random baselines. Evaluating against `GreedyStdAgent` rather than a random agent alone gives a much more informative picture; `KotaAgent` remained ahead in every run against this stronger opponent.

Single runs are noisy. Run-to-run standard deviation (≈ 0.08) was larger than several of the gaps between ablation configurations, so component effects must be confirmed across multiple runs rather than from a single tournament.

Independent adaptation improves performance. Separate buy and sell fractions allow better calibration than a shared parameter.

Conclusions

`KotaAgent` achieved competitive negotiation performance by combining profit margin constraints, adaptive pricing, fallback purchasing, and delivery risk management. Against the stronger `GreedyStdAgent` baseline it won every run, and a dedicated multi-run study showed that fallback buying improves the worst-case outcome without reducing average profit. Remaining work includes reporting per-world worst-case scores, tuning the delivery window, and handling within-step simultaneity with an explicit inventory account.