Computers and Artificial Intelligence; Vol.3, No.6; 2026
E-ISSN: 3006-3728 P-ISSN: 3006-371X
DOI: https://doi.org/10.70267/cai.v3n6.297313
Published by: Zeus Press

Marginal-Gain Candidate Scheduling for Low-Resolution QR Code Decoding Enhancement

Linhao Wang*

Henan Polytechnic University, Jiaozuo, Henan, China

**Corresponding author: Linhao Wang.*

Abstract

Low-resolution QR codes often cannot be decoded, and the usual fix—running many enhancement candidates and retrying wastes compute because different candidates keep rescuing the same images. To address these issues, we propose marginal-gain candidate scheduling (MGCS). MGCS uses a pool of 11 candidate configurations (scaling, contrast enhancement like CLAHE, sharpening, border padding, color inversion, and serial combinations). During the offline phase, a greedy selection rule on a development set determines both the retained candidates and their execution order. At each step, the chosen candidate is one of the highest score, defined as the number of newly decoded images divided by its average runtime. In the online phase, this pre-determined sequence is executed, and the process terminates immediately upon a successful decode. On an independent test set of 2,000 synthetically degraded images (cropped with ground-truth boxes), the 5-stage setup decodes 63.95% of samples in 1.36 ms per crop and the 10-stage setup reaches 65.50% in 3.13 ms—+2.80 and +4.35 points over a magnification-only retry. Against a fixed multi-scale retry, 5 stages cut average time about 22% at no significant accuracy change, while 10 stages add 2.30 points. A second, separately generated test set shows the same pattern. This difference is crucial, because candidate selection dictates higher accuracy. Conversely, execution order only impacts the runtime of a fixed set. Because every step is standard image processing, the method needs no extra trained restoration model and stays runnable on a CPU.

Keywords

QR code decoding, low resolution, marginal gain, candidate scheduling, decoding enhancement

1. Introduction

QR codes have become indispensable in applications such as robotic sorting, warehouse handling, and delivery by enabling information transmission and target identification. Their decoding results directly affect subsequent task execution [1,3]. However, in practical scenarios, factors such as shooting distance, motion, and imaging conditions may cause QR codes to occupy only a small number of pixels in images. Such conditions degrade module edge information through undersampling, blurring, and noise, thereby rendering conventional decoders ineffective. Robotic platforms usually lack powerful hardware and require fast responses, meaning we cannot indefinitely increase image processing costs. Improving the decoding success rate of low-resolution QR codes under strict computing budgets budgets is still very difficult.

Current literature predominantly addresses complex imaging conditions by improving object detection and image restoration, and by enhancing the decoding process. Detection methods refine target extraction and

© 2026 The Author(s). Authors retain copyright in their articles.

geometric correction to locate QR codes [1,6]. However, these detectors do not work when an image is correctly located but remains undecodable due to low resolution. Image enhancement techniques improve compatibility by using methods such as interpolation-based scaling, grayscale adjustment, and multi-version decoding [7,9]. Recently, researchers have applied learning-based super-resolution methods to make QR codes more readable [10,11]. However, these methods demand extra model training and longer inference time. In contrast, multi-candidate methods based on conventional image processing do not need extra learning models. They are simple to implement and easy to deploy on CPUs, making them highly practical for resource-constrained environments.

However, more candidates do not necessarily yield higher decoding efficiency. Image processing configurations vary in their adaptability to degraded images, and the successfully decoded samples may either complement each other or exhibit significant overlap. A new candidate contributes very little if it only decodes samples that previous ones got right. Instead, it needlessly consumes computational resources. Moreover, the execution time for different candidate configurations (e.g., scaling, contrast enhancement, and sharpening) varies significantly. If we rely only on single success rates or a fixed retry order, balancing decoding gains against computing costs becomes difficult. Effective multi-candidate enhancement thus requires building and scheduling configurations, while simultaneously accounting for both sample overlap and per-candidate cost.

Inspired by algorithm selection and combination [17], we propose a marginal-gain candidate scheduling (MGCS) method for low-resolution QR codes. Our method builds candidate configurations from various image enhancement and scaling operations. We evaluate how each candidate evaluate its marginal contribution—the number of new correct decodes relative to the selected set. We then establish a greedy criterion using the ratio of this marginal gain to the average execution time. In the offline phase, the execution sequence of candidates is determined using a development set; in the online phase, we perform enhancements and decoding sequentially according to the predetermined order. The process halts immediately upon receiving a valid result. Adjusting the number of candidate execution levels lets us trade off decoding success rate against computational latency. We focus only on the QR code cropping region and leave the object detector untouched detector. The equivalent module pixel density serves solely for offline analysis only and does not affect in online decision-making.

This work makes three main contributions:

(1) We develop a marginal-gain scheduling scheme that mitigates sample overlap and varying time costs in conventional multi-candidate decoding. This evaluation determines the execution sequence and enables decoding enhancement without relying on learned restoration models.

(2) We set up a controlled test to dissect how marginal-gain selection affects the decoding rate, and how execution order influences computational time, thereby validating the practical effectiveness of the framework.

(3) We benchmark our method against original decoding, magnification-only retry, and fixed multi-scale retry on an independently synthesized degradation test set. We analyze the performance trade-offs and boundaries of different configurations by combining equivalent-module-pixel-density stratification with a small set of real-world captured samples.

Table 1 compares the research scope of this work with existing related work.

Table 1: Comparison of positioning with existing work

Work	Multi-candidate retry	Candidate subset selection	Considers execution cost	Requires additional model training	Reports decoding time	References
References [6,8]	Decode after detection/correction	No	—	Not reported	No	Not reported
Reference [7]	Detection + perspective correction + multi-version decoding	Yes	Not reported	Not reported	The detector needs to be trained	Not reported
Reference [9]	Blurred-kernel estimation + restoration + decoding iteration	No (Single candidate iteration)	No	Not reported	No	Not reported

© 2026 The Author(s). Authors retain copyright in their articles.

Work	Multi-candidate retry	Candidate subset selection	Considers execution cost	Requires additional model training	Reports decoding time	References
Reference [10]	Adaptive multi-step super-resolution	No	—	Not reported	Yes	Yes (including PPM analysis)
Reference [11]	Deep super-resolution reconstruction	No	—	Not reported	Yes	Not reported
Reference [4]	YOLOv8 detection + decoding	No	—	Not reported	The detector needs to be trained	Not reported
This paper	Selection and scheduling of candidate processing configurations	Yes	Yes (marginal gain)	Yes (average time-in criterion)	No (do not train a restoration model)	Yes (mean and P90)

Note: "Not reported" indicates that the corresponding information was not provided in the public abstract or main text of the cited literature, and does not imply that the method inherently disregards this factor; "candidate subset selection" refers to whether the candidate subset is determined based on the complementarity of successful samples among candidates.

2. Marginal-Gain Candidate Scheduling Method

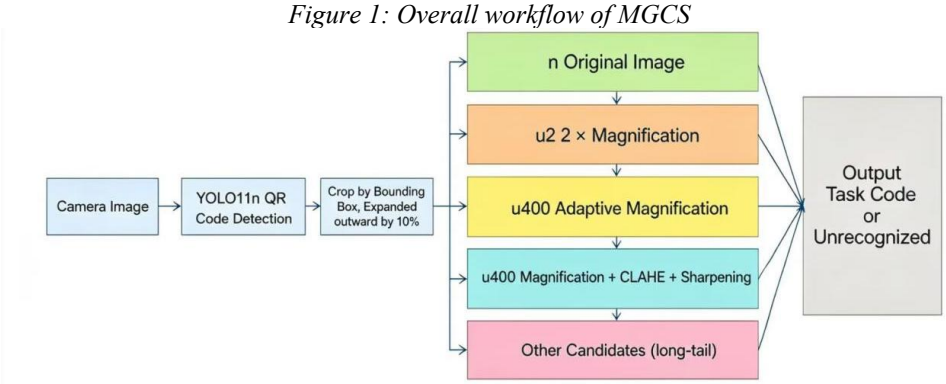
2.1 Problem Definition and Overall Process

We focus on enhancing low-resolution decoding for cropped QR codes. The system takes a cropped QR code block I_B as input and outputs either the decoded text or an "unrecognized" status. The cropping region typically comes from an object detector or manual labeling. In this study, we expand the bounding box by 10% to obtain the cropped block to preserve image information near the QR code boundaries. Object detection and candidate region generation are not part of the proposed MGCS algorithm.

For a given cropped block, while various image processing configurations may yield different decoding results, they often successfully decode the same overlapping samples and require varying amounts of time. Adding more candidates does not guarantee a proportional rise in success compared with the extra time spent. We thus frame low-resolution QR code enhancement as a subset selection and ordering problem. By measuring each candidate's extra decoding capability and time cost, we schedule those with higher marginal gains first.

We therefore propose the Marginal-Gain Candidate Scheduling (MGCS) method. The framework is split into two phases: offline candidate selection and online sequential decoding. In the offline stage, candidate configurations are evaluated on the development set based on their correct decoding rates and average processing time. A greedy algorithm using the marginal-gain criterion determines the execution sequence of candidates, which is then fixed before testing and deployment. During the online stage, we run the candidates one by one following the fixed sequence. We terminate the process immediately when the decoder returns a valid result. If no candidate yields a valid output, the system outputs an "unrecognized" status; the overall workflow is illustrated in Figure 1.

Figure 1: Overall workflow of MGCS



© 2026 The Author(s). Authors retain copyright in their articles.

2.2 Candidate Pool Construction

Low-resolution QR codes vary in scale, contrast, edge sharpness, and crop boundaries. To cope with this issue, we construct a candidate pool of 11 image processing configurations (Table 2). This group features original decoding, fixed-scale resizing, adaptive magnification, contrast enhancement, sharpening, boundary padding, color inversion, and several serial combinations.Each configuration applies predefined parameters independently to the cropped block; they do not share intermediate results.

Table 2: Candidate pool (11 candidate processing configurations)

ID	Candidate	Transformation and parameters
n	Original image	Decode directly without image enhancement
u2/u4/u6	Fixed magnification 2×/4×/6×	Bicubic interpolation with magnification factors of 2, 4, and 6
u400	Adaptive magnification	Targeting a short side of approximately 400 pixels, with a magnification limit of 1–6
u400c	Adaptive magnification + CLAHE	Adaptive magnification, grayscale conversion, and CLAHE enhancement
u400s	Adaptive magnification + sharpening	Adaptive magnification and unsharp-mask enhancement
u400cs	Adaptive magnification + CLAHE + sharpening	Adaptive magnification, CLAHE enhancement, and sharpening
u2c	2× magnification + CLAHE	Fixed 2× magnification, grayscale conversion, and CLAHE enhancement
pad	Border padding + global histogram binarization	Pad the edges according to the crop-block size and use global histogram-based binary decoding
inv	Color inversion	Decode after inverting the BGR three-channel colors

We designed the candidate pool by considering how different image processing operations affect the decoder. For instance, interpolation-based magnification alters the pixel scale of QR modules, while contrast enhancement and sharpening modify the grayscale distribution near edges. Additionally, boundary padding adjusts the background range, and color inversion changes the grayscale polarity. Different processing methods may improve decoding results under various degradation conditions. Such variations induce complementarity among correctly decoded samples.

Although these steps do not recover lost data, they merely reformat the image for the decoder. Since different candidates often decode the same images, simply adding more candidates will not yield a continuous improvement in success rate. Instead of assuming any single enhancement is universally effective, we evaluate each candidate's decoding performance and computational cost on the development set. We then select and rank candidates based on the marginal-gain criterion described in Section 2.3. The candidate pool provides a diverse configuration space, and the core improvement of MGCS lies in selecting candidates from this pool and determining their order.

2.3 Offline Selection and Online Execution

To mitigate redundant computations arising from overlapping successes, a greedy selection approach is employed to determine the execution sequence based on the newly added correct decodes and the computational cost of each candidate.

Let the development set be

$$D = \left\{ \left(I_i , y_i \right) \right\}_{i=1}^{N} \tag{1}$$

Where I_i is the cropped image of the *i* QR code and y_i is the corresponding actual encoded content; the candidate pool is denoted as P. For any candidate configuration c, let its corresponding image transformation be denoted as T_c, and the set of correctly decoded samples is defined as

$$S_c = \left\{ i \mid Decode\left(T_c \left(I_i \right) \right) = y_i \right\} \tag{2}$$

© 2026 The Author(s). Authors retain copyright in their articles.

Which represents the set of samples whose decoded results exactly correspond to the true labels after processing by the candidate. Let C be the current selected candidate set, and let the union of samples correctly decoded by the selected candidates be

$$S_{sel} = \bigcup_{j \in C} S_j \tag{3}$$

The marginal increase in success count of candidate *c* relative to the current selected set is defined as:

$$\Delta(c \mid C) = \left| S_c \setminus S_{sel} \right| \tag{4}$$

$\Delta(c \mid C)$ represents the number of new samples candidate c can decode correctly that have not been covered by the selected candidates. Unlike evaluation methods based solely on a candidate's own decoding success rate, this metric accounts for the overlap of successful samples among candidates. It thus measures the actual contribution of a new candidate to the overall decoding coverage.

To account for varying computational costs, we incorporate the average execution time t_c into our criterion. This lets us trade off marginal gain against computational cost:

$$c^* = \arg\max_{c \in P \setminus C} \frac{\Delta(c \mid C)}{t_c} \tag{5}$$

Here t_c is the average execution time for image transformation and decoding for candidate c across all development samples, including failed decoding attempts. Our criterion balances extra correct decodes against computational cost, thereby avoiding the pitfall of ignoring sample overlap when sorting solely by individual success rates. Because this is a greedy selection approach, it relies on development-set statistics and may not find the globally optimal sequence.

The offline selection process begins with the computation of correct decoding sets and average execution times for all pool candidates. An empty selected set is then initialized, and the candidate maximizing Equation (5) is iteratively chosen. This process updates the covered samples until all candidates are ranked. We place candidates with zero marginal gain toward the end of the sequence. If multiple candidates yield identical scores, we prioritize the one with lower average processing time, and each candidate is processed independently without sharing intermediate results.

To accommodate varying computational budgets, we derive two decoding enhancement configurations (5-stage and 10-stage) from the same candidate sequence. The specific selection criteria are determined by combining the accurate decoding rate and average processing time on the development set, as detailed in Section 4.1. We perform both candidate selection and sequence determination on the development set, completely excluding the test set from ranking and configuration selection.

During the online phase, no score calculations are performed; instead, image processing and decoding are applied sequentially based on the fixed offline order. The process halts right away and return it as soon as the decoder returns valid content. If none of the candidates return valid content, "unrecognized" is output. Since the real payload content is unavailable during the online phase, early termination is based on the decoder's output rather than the actual label; the final decoding decoding rate is evaluated by comparing the decoded outputs against real labels.

Candidate picking candidates and running them order affect final performance in distinct ways. Assuming zero false-positive decodes, the correctly decoded sample set relies entirely on the union of individual candidate success sets. If we change the execution order, the set of decodable samples stays the same. However, the average number of steps and the total computing time will change. Candidate subset selection primarily determines the range of correctly decoded samples that can be covered, while execution order mainly influences the processing cost under early-termination mechanisms. Section 4.2 conducts separate tests comparing various candidate subsets and reordering inside a predetermined group to validate these effects.

© 2026 The Author(s). Authors retain copyright in their articles.

Table 3: Marginal-gain candidate choosing and running

Stage	Operation
Input	Development set D, Candidate pool P; Online cropped block I_B; Execution level K.
Offline 1	Compute the correct-decoding sample set S_c and average processing time t_c for each candidate c.
Offline 2	Start with an empty selected set $C = \varnothing$, the coverage set $S_{sel} = \varnothing$ and the candidate sequence.
Offline 3	Pick the candidate c^* having the highest marginal-gain score per Equation (5), append into the sequence L, and update C and S_{sel}
Offline 4	Repeat Offline 3 until all candidates are ranked; freeze the sequence L and take its prefix of length K (K = 5 or 10)
Online 1	Execute the K-th candidate configuration and call the decoder
Online 2	If valid decoded content is returned, output it immediately and terminate; otherwise move on to the next candidate
Online 3	If none of the candidates return a valid result, output "unrecognized"

3. Experimental Setup

3.1 Dataset and Degradation Setup

We built our dataset with a synthesizer where all parameters can be controlled. This let us test MGCS thoroughly under many different low-resolution conditions and image degradations. Our synthetic images give precise ground truth and precise control over resolution and degradation. Consequently, we can easily collect correctly decoded samples, analyze candidate complementarity, and compare different decoding approaches.

QR codes compliant with the standard [14] were generated. These use versions 1 to 8, with error-correction levels distributed as L (12%), M (52%), Q (24%), and H (12%). Payload sizes range from 6 to 130 bytes. Various imaging conditions were then simulated via geometric projection and degradation operations, including perspective transformation, motion blur, Gaussian noise, changes in brightness and contrast, occlusion, JPEG compression, specular highlights, and slight defocus. Perspective transformation is simulated using pinhole projection and homography. Motion-blur length is normalized by module width. To simulate limited resolution, we perform scaling via 12× supersampling followed by area-averaged downsampling. The specific parameter ranges for each degradation type are shown in Table 4, and typical examples are illustrated in Figure 2.

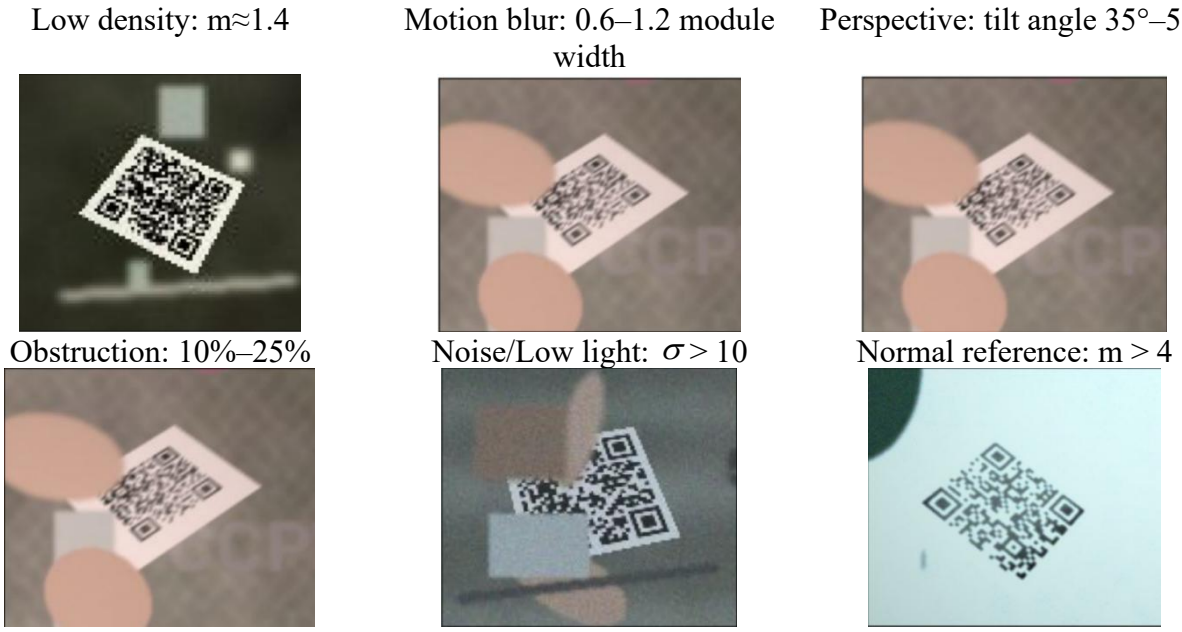
Table 4: Degradation modeling and parameter ranges

Degradation	Implementation method	Recorded parameter	Parameter range
Perspective transformation	Pinhole projection and homography transformation	Angle formed by the normal and the optical axis	0–60°
Motion blur	Linear kernel, length normalized by module width	blur_modules	0–1.2
Gaussian noise	Pixel-wise brightness noise	noise_sigma	0–22
Brightness / Contrast	Gain, bias, gamma, color shift, and vignetting	Relative gain	0.5–1.6
Zoom	Downsampling after 12× supersampling rendering	Equivalent module pixel density	1.3–6.6
Obstruction	Random color-block overlay	Coverage	0–25%
Imaging and compression	JPEG, specular highlights, and slight defocus	JPEG quality factor	65–95

Since QR code versions vary in module count, relying only on total pixel width fails to capture the sampling condition of individual modules. We therefore adopt equivalent module pixel density for offline hierarchical analysis. We define this metric as the ratio of the average marked-area side length divided by the number of modules. It is useful for comparing correct decoding rates at various sampling scales but does not affect online candidate selection.

© 2026 The Author(s). Authors retain copyright in their articles.

Figure 2: Examples of mixed degradation (each grid is dominated by one degradation type; parameter ranges in Table 4)



The module pixel density recorded by our synthesizer slightly differs from the recalculated density (based on the final bounding box) due to differences in computation steps. This causes approximately 1% to 2% cross-bin migration for some samples. Furthermore, the equivalent module pixel density is an approximate metric at the overall scale and cannot fully capture how sampling changes in different parts of the QR code when perspective distortion is present. The density of real-world samples is estimated using the ratio of effective code width to module count. This method differs from the synthetic approach and serves only as a rough reference.

To prevent candidate selection from biasing the test results, the data was split into six distinct subsets. These include a training set, a development set, an assistance set, main dataset A, validation set B, and a stress set. The sample sizes and purposes of each set are shown in Table 5. Both the development set and the two test sets are constructed using joint stratification based on equivalent module pixel density and perspective tilt angle, with each predefined stratum generating 100 images, and all sets contain non-overlapping encoded content.

Table 5: Role allocation of the dataset

Set	Scale	Purpose	Random seed	Affects final algorithm
Detection training set	16,000 + 2,000 (verification)	Train the YOLO11n detector	20260101	Trains only the external detector, not involved in candidate reward evaluation
Development set	2,000	Determine candidate sequences and two-tier configurations	20260101	Yes
Selection-assistance set	2,000	Exploratory comparisons during early method refinement, not for final reporting	777777	Indirect (early stage)
Main dataset A	2,000	Report the main performance results	888888	No
Validation test set B	2,000	Reproduction and verification	999999	No
Stress set	2,000	Boundary analysis under stronger degradation	20260101	No

Dataset A and validation set B were kept strictly separate from candidate pool construction and sequence selection. Since all datasets share the same synthesizer, our results show performance on unseen samples from the same generation process, rather than cross-domain generalization.

© 2026 The Author(s). Authors retain copyright in their articles.

The ground-truth encoding of the dataset was directly recorded during the synthesis process. To verify the consistency between the images and their recorded labels, we randomly selected 300 images for supplementary decoding. Among them, 210 images decoded successfully and perfectly aligned with the recorded labels. The remaining 90 images still failed to decode. The above inspection did not reveal any inconsistencies in the successfully verified samples, but this does not prove that all non-decodable samples have had their labels independently validated.

3.2 Comparison of Strategies and Configurations

We evaluated four decoding approaches: original decoding, magnification-only retry, fixed multi-scale retry, and MGCS.

Original decoding directly feeds the cropped block to the decoder without any enhancement. This establishes the baseline decoding capability.

For magnification-only retry, if original decoding fails, we apply adaptive magnification and call the decoder again. This evaluates the improvement from basic scale adjustment and secondary decoding.

The fixed multi-scale retry method applies 2x, 4x, and 6x scaling sequentially if the initial decoding fails. The process stops at the first valid result. We use this to evaluate fixed candidate combinations and a predefined order.

For MGCS, we apply the marginal-gain scheduling method proposed in Section 2. After determining the candidate sequences on the development set, MGCS selects the 5-stage and 10-stage configurations. These two configurations share the same criteria and execution order; they differ only in the number of executed levels. This allows us to analyze the trade-off between decoding rate and processing time.

To ensure a consistent comparison, all strategies use the same input cropping blocks, crop-expansion ratio, decoder, and hardware environment, and follow the same "early termination upon valid result" rule. For the main performance comparisons, crops are generated using ground-truth bounding boxes to eliminate interference from localization errors.

These comparisons warrant careful interpretation. The 5-stage MGCS configuration consists of two candidates: original decoding and adaptive magnification. Its improvement over magnification-only retry reflects the gain from expanding the candidate set. However, this alone cannot prove that our marginal-gain criterion outperforms other selection rules. Similarly, MGCS (10-stage) and fixed multi-scale retry (4 candidates) differ in both candidate composition and quantity; thus, the overall performance comparison primarily reflects the practical trade-off between correct decoding rate and computational time under various decoding methods. To further validate the selection criteria, Section 4.2 establishes a controlled experiment with an equal number of candidates but different selection rules. It distinguishes the impact of candidate subset selection from execution order by fixing the candidate set and varying only the execution sequence.

All candidate configurations and execution sequences were finalized prior to the main test, with Dataset A and validation set B playing no role in selecting the MGCS sequence. The corresponding datasets, sources of cropping boxes, and whether detection time was included for each experiment are shown in Table 6.

Table 6: Experimental settings for each result table

Result table	Set	Crop-box source	Decoding method	Includes detection time
Table 7	Development set	Bounding box	Greedy-selection order (offline)	No
Table 8, Table 9(a)	Main dataset A	Bounding box	See table	No
Table 9(b)(c)	Development set	Bounding box	See table	No
Table 10	Main dataset A	Bounding box	MGCS (10-stage)	No
Table 11	Stress set	Detection box	MGCS (10-stage)	No
Table 12	Development set	Bounding box	Fixed candidate combination (control)	No
Table 13, Table 14	Real-world samples	Detection box	MGCS (10-stage)	No

© 2026 The Author(s). Authors retain copyright in their articles.

3.3 Evaluation Metrics and Operating Environment

The MGCS method is evaluated from three perspectives: correct decoding rate, computational time, and comparative performance across various approaches.

The correct decoding rate (CDR) is defined as the number of correctly decoded samples divided by the total number of test samples:

$$R_{CDR} = \frac{N_{correct}}{N_{total}} \times 100\% \tag{6}$$

where $N_{correct}$ represents the number of correctly decoded samples and N_{total} represents the total number of samples involved in evaluation. Sample outcomes are grouped into three categories: exact matches to ground truth (correct), mismatches (incorrect), and instances where no valid candidate emerges (decoding failure). The latter two cases are not treated as correct decodings. To calculate the hierarchical correct decoding rate, we divide the number of correctly decoded samples by the total samples in that specific layer. Interestingly, across the 2,000 test samples, no valid outputs mismatched the ground truth. However, this empirical observation should not be interpreted as a zero probability of erroneous returns in general.

Computation time serves as a metric to assess the execution cost of various decoding methods. We measure the time from when the cropped block enters the decoding pipeline until the algorithm returns a result. This includes candidate image transformations and decoder invocations, but excludes object-detection time required for inference. The timing statistics include both correctly and incorrectly decoded samples, without excluding failed cases. We report both the average decoding time and the 90th-percentile (P90) time. These metrics capture the mean processing cost and the performance on relatively slower samples, respectively. P90 does not represent the worst-case latency. Meeting an average time budget does not guarantee all images will finish by the same deadline. Two types of timing should be distinguished: the offline cost estimation t_c in Section 2 for the candidate-selection criterion, which assumes independent execution of candidates and is used to determine the sequence; and the online timing evaluated here, measured based on actual candidate sequences and early-termination processes. Because these two metrics come from different sample sets and timing conditions, their numerical values do not need to match.

To assess the statistical significance of decoding rate differences, we adopted a paired comparison approach. We record the decoding results for each image under both strategies and categorize the samples into four groups: correctly decoded by both strategies, correctly decoded only by the baseline method, correctly decoded only by the proposed method, and incorrectly decoded by both. We use McNemar's exact test to assess the paired differences. We also apply a bootstrap method with 3,000 image-level paired resamplings to estimate the 95% confidence interval of the decoding rate difference. In each resampling, we preserve the paired relationship by sampling the results of both strategies for the same image. A non-significant difference does not imply equivalence between the two strategies. Proving equivalence would require predefined margins and formal equivalence testing.

We use zxing-cpp 3.1.1 as our primary decoder. For supplementary comparisons, we also employ OpenCV's QRCodeDetector and QRCodeDetectorAruco. The object-detection module adopts YOLO11n with an input size of 640×640, a batch size of 16, and 30 training epochs; the model from epoch 26 is used for detection experiments.

The experimental hardware setup includes an Intel Core i7-13650HX processor and an NVIDIA RTX 5060 Laptop 8GB graphics card, while the software environment consists of Windows 11 and Python 3.10. We measure the decoding-enhancement times in Section 4 on the CPU. These times do not include YOLO11n detection inference, as we use the GPU solely for detector training and inference. Decoding timing follows a single-threaded procedure with a 20-frame warm-up before each segment. All decoding strategies are evaluated under identical hardware and testing conditions.

© 2026 The Author(s). Authors retain copyright in their articles.

4. Experimental Results and Analysis

4.1 Configuration Determination and Overall Performance

We used Equation (5) to compute the marginal additional success count and average execution runtime of every candidate on the 2,000 development set images. We then determined the candidate execution sequence using a greedy strategy. The results are shown in Table 7. Here, marginal success count is the number of new samples a candidate decodes correctly that previous candidates missed. We define the candidate score as this count divided by the average execution time.

Table 7: Greedy sequence on the development set (n = 2,000)

Samples	Candidate	Marginal incremental success count (images)	Cumulative CDR	Single-candidate processing time (ms)	Score (images/ms)
1	Original image n	897	44.85%	0.163	5503.1
2	u2 fixed 2× magnification	+297	59.70%	0.499	595.2
3	u400 adaptive magnification	+115	65.45%	1.359	84.6
4	u400cs magnification + CLAHE + sharpening	+39	67.40%	1.477	26.4
5	u2c 2× magnification + CLAHE	+6	67.70%	0.688	8.7
6	inv color inversion	+1	67.75%	0.134	7.5
7	u400s magnification + sharpening	+13	68.40%	1.828	7.1
8	pad border padding	+1	68.45%	0.146	6.8
9	u400c magnification + CLAHE	+4	68.65%	1.190	3.4
10	u6 fixed 6× magnification	+9	69.10%	3.867	2.3

Note: the time listed represents the average development-set execution runtime of every candidate when run individually; scores are computed from these times (time rounded to three decimal places, score approximated to one decimal place), with the original-image row scoring 897 images per 0.163 ms. Table 7 shows that 6-stage (color inversion at 7.5 images/ms) outperforms 7-stage (magnification and sharpening at 7.1 images/ms), confirming our selection criterion.

Original decoding achieved 44.85% accuracy on the development set. This increased to 67.40% with 4-stage candidates, 67.70% with 5 stages, and 69.10% with 10 stages. The first few candidates contribute most to the accuracy improvements. Meanwhile, subsequent additions give us fewer and fewer new decodings. While expanding the candidate pool does increase coverage, this incremental gain must be weighed against the extra processing cost.

We selected the 5-stage and 10-stage settings according to their cumulative decoding rates and average times. These two settings constitute our MGCS configurations: the 5-stage configuration focuses on controlling average decoding time, while the 10-stage configuration achieves a higher correct decoding rate by increasing candidate attempts. Both configurations share the same candidate sequence; they differ only in the number of executed levels and were fixed before testing on the evaluation set.

Table 8 lists the performance results for dataset A. The baseline decoding achieved a correct decoding rate of 42.75%, while magnification-only retry and fixed multi-scale retry improved it to 61.15% and 63.20%, respectively. MGCS with the 5-stage and 10-stage configurations achieved correct decoding rates of 63.95% and 65.50%, representing improvements of 2.80 and 4.35 percentage points over magnification-only retry, with corresponding average CPU decoding times of 1.36 ms and 3.13 ms.

The 5-stage MGCS increased the accuracy by 0.75 percentage points over fixed multi-scale retry, though this difference is not statistically significant. However, it cut the average time from 1.75 ms to 1.36 ms (about 22%).For the 10-stage configuration, MGCS achieved a 2.30-percentage-point increase in correct decoding rate, and this improvement was statistically significant, but the average processing time increased to 3.13 ms

© 2026 The Author(s). Authors retain copyright in their articles.

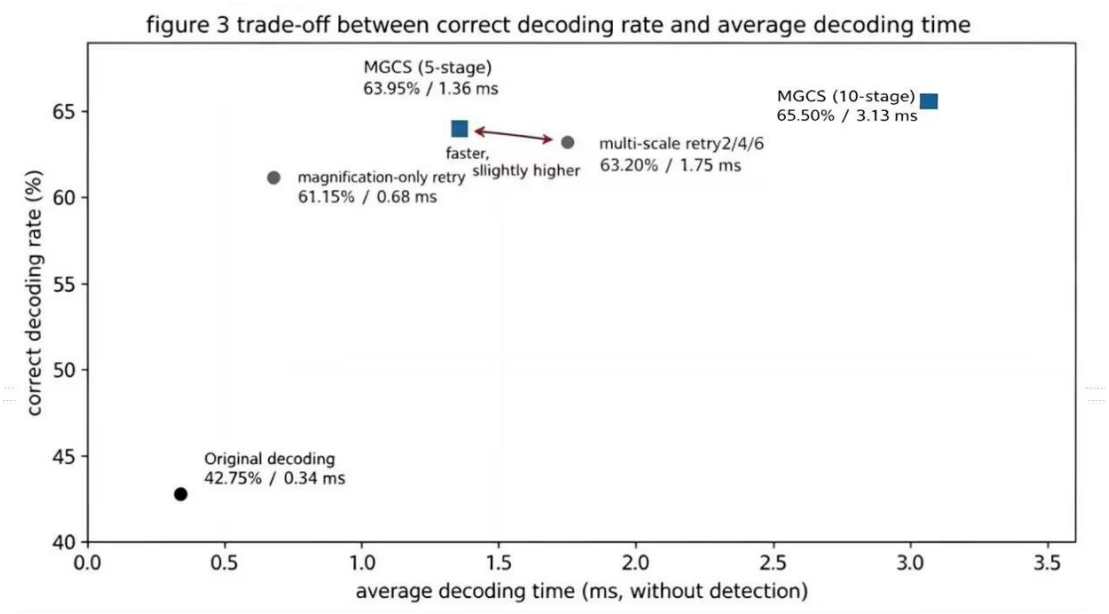
(about 79% higher than 1.75 ms). These results indicate the 5-stage setup is more efficient when computing power is limited, whereas the 10-stage variant boosts accuracy but sacrifices longer execution time.

Table 8: Overall performance (main dataset A, n = 2,000, bounding box, without detection)

Strategy	Candidate numbers	Correctly decoded images	Correct decoding rate	Average decoding time	P90
Original decoding	1	855	42.75%	0.34 ms	0.69 ms
magnification-only retry	2	1223	61.15%	0.68 ms	1.32 ms
Fixed multi-scale retry (2×/4×/6×)	4	1264	63.20%	1.75 ms	4.29 ms
MGCS (5-stage)	5	1279	63.95%	1.36 ms	3.32 ms
MGCS (10-stage)	10	1310	65.50%	3.13 ms	8.57 ms

Validation Set B, completely excluded from the selection phase, yielded correct decoding rates of 65.55% and 67.70%, respectively—representing improvements of 2.85 and 5.00 percentage points over magnification-only retry. Compared to fixed multi-scale retry, the accurate decoding rate for the 5-stage remained nearly unchanged, while that for the 10-stage increased by 2.00 percentage points. The consistent trends between both test sets indicate that the candidate the order stays mostly the same for samples from the same synthetic framework. The success-rate-versus-time relationship is shown in Figure 3.

Figure 3: Trade-off between correct decoding rate and average decoding time (main dataset A, n = 2,000, without detection)



Because candidate composition and count vary across approaches, our main goal is to assess the trade-off between decoding capability and processing cost. As a result, an exact match with the ground truth cannot be guaranteed, and such discrepancies warrant further work. To investigate whether this gain stems from marginal-gain selection, we next compare candidate subsets and reorder fixed sets to isolate the effects of picking candidates and running them order.

4.2 Candidate Selection and Execution Order Analysis

MGCS involves both candidate subset picking candidates and running them-order determination. We conducted tests comparing various candidate subsets and reordering fixed sets to clarify how these factors affect decoding. We also included paired comparisons to the baseline to analyze the gains in correct decoding rate and changes in computational time, as shown in Table 9.

We first assessed how the marginal-gain criterion influences candidate subsets. We compared two selection methods using identical candidate sets and counts. The first uses our marginal-gain criterion; the second ranks candidates solely by their individual success-rate-to-time ratio. Both selected five configurations. On the

© 2026 The Author(s). Authors retain copyright in their articles.

development set, these two methods achieved 67.70% and 61.20%, respectively. This results in a 6.50 percentage percentage point increase in marginal-gain selection. To reduce potential bias from development-set selection, both candidate subsets were frozen and subsequently validated on set A and validation test set B.

Table 9: Decomposition of candidate choosing and running order ((a) main dataset A; (b)(c) development set)

Group	Comparison	Result
(a) Relative to magnification-only retry	5-stage / 10-stage	5-stage + 2.80 pp (56 gained / 0 lost, $p = 2.8\times10^{-17}$); 10-stage + 4.35 pp (87 gained / 0 lost, $p = 1.3\times10^{-26}$)
(a) Relative to multi-scale retry	5-stage / 10-stage	5-stage + 0.75 pp (34 / 19, $p = 0.053$); 10-stage + 2.30 pp (50 / 4, $p = 3.8\times10^{-11}$)
(b) Same set of 5 candidates, order changed	Greedy / ascending-time; Random / descending-time	CDR = 67.70% in all cases; average processing time 1.22/1.23/1.31–1.34/1.92 ms
(c) Same number of candidates (5 each), different selection rules	Marginal-gain selection; Sort by single-level success rate divided by time taken	67.70%/61.20% (difference of 6.50 pp, 95% CI [5.45,7.60])

For dataset A, marginal-gain selection achieved 63.95% compared to 57.70% for single-stage-efficiency selection. This is a 6.25-point improvement (95% CI: [5.20, 7.35]). On validation Set B, the rates were 65.55% and 59.60%, respectively, a 5.95-point increase (95% CI: [4.90, 7.05]). Across both test sets, marginal-gain selection consistently achieved covers more correctly decoded samples. This suggests overlapping successes introduce redundancy. By accounting for previously decoded samples, marginal-gain selection allows a smaller set of candidates to cover more unique decodable samples. Ultimately, the superior decoding rate of MGCS in our setup stems primarily from careful candidate subset selection, instead of just adding more candidate count.

Next, we analyze how execution order affects computational time. We fixed the candidate sets to those of the 10-stage MGCS configuration and tested four execution orders: marginal-gain greedy order, ascending time-cost order, random order, and descending time-cost order. The results show that all four orders achieved a correct decoding rate of 67.70%, but their average processing times were 1.22 ms, 1.23 ms, 1.31–1.34 ms, and 1.92 ms, respectively. Among them, the marginal-gain greedy order had the lowest average time, yet the difference compared to the ascending time-cost order was only about 0.01 ms. In this experiment, execution order mainly affects processing cost under early termination, whereas the composition of the candidate set directly influences correct-decoding coverage. We do not assert that greedy ordering significantly outperforms simple ascending time-cost ordering.

Assuming no erroneous decodes occur, the overall accurate coverage of a fixed candidate set is the same as the union of each candidate's performance sets. The execution sequence has no impact this coverage but might affect the actual number of candidates executed for different samples. This reordering experiment aligns with our theoretical analysis. Practically, decoding processes where incorrect content might be returned early, the final decoding decoding rate should still be based on complete online-execution results; this section only reports observations under the given experimental conditions.

Further analysis of the paired-comparison results in Table 9(a) shows that, compared to magnification-only retry, MGCS with the 5-stage and 10-stage configurations increased the number of correctly decoded samples for dataset A by 56 and 87, respectively, with no instances observed where only the baseline was correct. Compared to fixed multi-scale retry, 5-stage added 34 correct samples while losing 19, and 10-stage added 50 correct samples while losing only 4. Notably, the difference in correct decoding rate between 5-stage and fixed multi-scale retry did not reach statistical significance, whereas 10-stage showed a significant improvement.

In general, the performance advantage of MGCS originates primarily from subset selection based on marginal gains, which broadens correct-decoding coverage under a fixed candidate count. Execution order, by

© 2026 The Author(s). Authors retain copyright in their articles.

contrast, mainly controls processing cost under the online early-termination mechanism. Putting these two parts together gives us the candidate-scheduling scheme we propose.

4.3 Density Stratification and Applicable Boundaries

Table 10 presents the stratified results for dataset A based on equivalent module pixel density. The number of samples in each bin ranges from 398 to 401, as the bins are assigned based on measured equivalent density, while the synthesis process relies on module_px at rendering time for layering; the difference in measurement procedures leads to approximately 1%–2% cross-bin migration.

Table 10: Categorized by equivalent module pixel density (main dataset A, annotated boxes, n = 2,000)

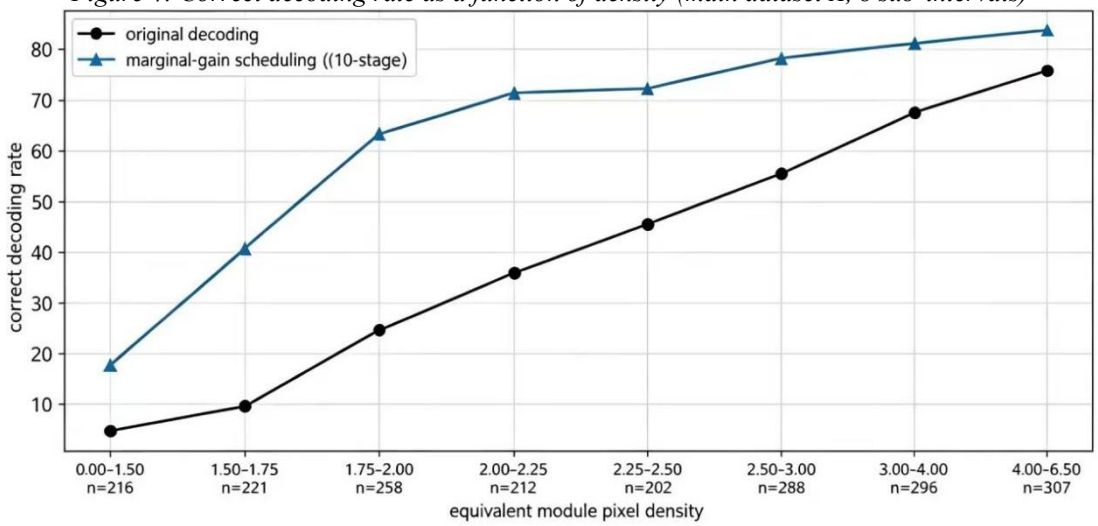
Density range	n	Original decoding	Only magnification and retry	Multi-scale retry	MGCS (10-stage)	MGCS−magnification-only
S1 [1.3,1.7)	400	27/6.8%	95/23.8%	101/25.2%	108/27.0%	+3.2
S2 [1.7,2.1)	398	102/25.6%	231/58.0%	242/60.8%	258/64.8%	+6.8
S3 [2.1,2.6)	401	176/43.9%	271/67.6%	280/69.8%	292/72.8%	+5.2
S4 [2.6,3.4)	400	249/62.3%	301/75.2%	312/78.0%	320/80.0%	+4.8
S5 [3.4,6.6)	401	301/75.1%	325/81.0%	329/82.0%	332/82.8%	+1.7
Overall	2000	855/42.75%	1223/61.15%	1264/63.20%	1310/65.50%	+4.35

Note: in the last column, MGCS − magnification-only is the per-category gain (exact number of MGCS − exact number of magnification-only retry) / category size; e.g., S5 = (332−325)/401 ≈ +1.7 percentage points.

Our data covers a range of 1.3 to 6.6 pixels per module. Compared to original decoding, gains are higher in tiers S1 to S3, partly due to magnification. The net gains over magnification-only retry are: S1 (+3.2), S2 (+6.8), S3 (+5.2), S4 (+4.8), and S5 (+1.7) percentage points. MGCS's marginal gain concentrates in the 1.7 to 2.6 pixels-per-module range. Higher gains appear in intermediate-density intervals, likely related to keeping information and baseline decoding capability. Since density was not varied in isolation from other degradation factors, no causal conclusions can be drawn. Figures 4 and 5 show how the accurate decoding rate varies with density,and how failed samples are composed.

Table 11 shows the results for the stress set (n = 2,000, detection boxes). Overall, the gain of the stress set decreased to +8.0 percentage points; among them, the number of correctly decoded samples with density ≤ 1.2 increased from 0 to 2 (1.3%). This subset also contains significant blur and noise, so the judgment that "failure is determined solely by density" only holds after controlling for other degradation factors. We merely observed that, for our tested degradation combinations, single-frame interpolation and conventional enhancement fail almost entirely at densities ≤ 1.2. In such regimes, adjusting the physical acquisition conditions is a prerequisite before applying any algorithmic enhancement.

Figure 4: Correct decoding rate as a function of density (main dataset A, 8 sub-intervals)



© 2026 The Author(s). Authors retain copyright in their articles.

Figure 5: Distribution of failed samples by predefined priority categories (main dataset A; original-decoding failure: 1,145 images; 10-stage configuration failure: 690 images)

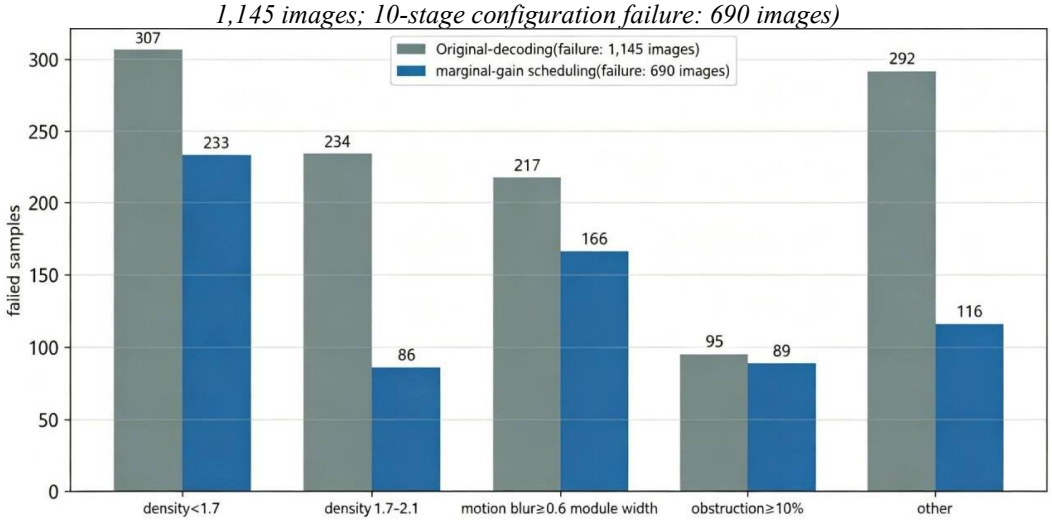


Table 11: Stress-set results (detection box, n = 2,000)

Subset	n	Original decoding	MGCS (10-stage)
All stress-set samples	2000	19.8%	27.8%
Density ≤ 1.2	159	0 images / 0.0%	2 images / 1.3%

Note: this table uses the same cropping protocol as Table 7. Since MGCS sequences were not rebuilt and re-evaluated specifically for each decoder, these results only indicate that the "resolution-normalization plus candidate-combination" method works across decoders, but do not completely prove the cross-decoder effectiveness of the MGCS scheduling strategy.

Table 12: Cross-decoder comparison of fixed candidate combinations (development set, annotated boxes, n = 2,000)

Decoder	Original decoding	+ Resolution normalization	+ Fixed candidate combination
zxing-cpp	44.9%	64.5%	67.2%
OpenCV QRCodeDetector	30.2%	44.8%	50.6%
OpenCV QRCodeDetectorAruco	24.3%	42.9%	49.3%

Table 12 uses the same cropping protocol as Table 7 (development set, annotated boxes, expanded by 10%). The original-decoding rate of zxing-cpp is 44.9%, which matches the 897/2,000 = 44.85% in Table 7 (a this small gap is purely a rounding effect).

4.4 On-site Observation and Discussion

Our real-world data comes in three batches, and they use different image sets and different denominators. Because of this, we cannot simply add their numbers together (Table 13): 13 images from early diagnosis, gathered while the method was still being built and used only to spot problems; 33 images collected later, fully separate from method development and the ones we report on; and 12 low-density images shot on an A4 printout from 2.9 to 7.1 meters with a 12MP phone, meant to push into the low-density range.

Reading the 33 later samples, the clean-condition ones measured roughly 2.0 to 10.4 pixels per module — squarely inside what plain decoding already handles. One sample near 1.4 pixels per module (a 29-pixel-wide code at 45 cm) sat right at the edge of the failure zone, where both our setup and the baseline came up empty. Back in early diagnosis we hit one telling case: the full image decoded fine, yet every cropped block failed. Enlargement, CLAHE and Otsu thresholding did nothing; it only read out once we padded white borders around the crop or switched to global histogram binarization. That single case points to crop boundaries and binarization settings as real causes of failure — though we only saw it once, so we treat it as a hint, not a finding.

The 12 low-density images are in Table 14. The crucial observation is that print size and distance are interchangeable: what decides how hard a code is to read is not how big it looks on the table, but rather how many pixels land on one module inside the image. Between the two printed batches, the fair thing to compare

© 2026 The Author(s). Authors retain copyright in their articles.

is that the equivalent density instead of the physical dimensions, which say little about the code's true size in the field.

Table 13: Define links and results of the real-world data

Set	images	Relationship	Result
Early-diagnosis images (for method standardization)	13	Concurrent with method development, not included in statistics	Identify failure phenomena related to cropping boundaries and binarization, leading to the generation of candidate white-border padding
Later independent collection (for reporting)	33	Independent of method development	Clean-condition samples mostly fall within the originally decodable range; among 33 cases, no additional successful configurations relative to original decoding were observed
Full-image reference sample	17	Different subset, denominator	1536 inputs + top-3 candidates can reach 16/17
Low-density supplementary	12	Independent collection	See the low-density section

Table 14: Low-density real-world results (12 images; density = code-area width / number of modules)

Detection & cropping configuration	Original decoding	MGCS (10-stage)	Instructions
First frame, expanded outward by 10%	3/12	3/12	Consistent with the default configuration of the synthesis experiment
First 3 frames, expanded outward by 50%	6/12	6/12	The number of candidate boxes increases with the expansion
First 10 frames, expanded outward by 50%	9/12	12/12	The number of candidate boxes continues to increase
First 10 frames, expanded outward by 10%	9/12	12/12	Fixed number of candidate boxes, comparing only the expansion ratio

Two key factors drive the results on the 12 images. First, how many candidate boxes the detector hands over matters most: plain decoding jumped from 3/12 to 9/12 just by taking more boxes. Letting the candidate configurations run over the top-10 boxes added three more (9/12 to 12/12). Second, once we pin the count at 10 boxes, expanding the crop by 10% or by 50% yields the gives the identical results. So, the expansion ratio did not expansion ratio did not affect the outcomes in this batch. Since our table does not test combinations of these factors, it is hard to separate which specific decoding operator drove the improvement in this batch. With only 12 images — 2 to 4 in each category — these figures describe this batch and nothing more; no statistical claim about real-world performance follows from them. Moreover, with no magnification-only baseline on this set, so we cannot determine whether any real-world gain comes from subset selection.

MGCS works best when the crop region is pre-provided, localization is reliable, and no learned restoration model is available. Under such circumstances, when the capture setup is fixed by the application, the payoff is a higher success rate for a cost evaluated in milliseconds on a CPU. If the task lets you move the camera instead, fix it at the source first — stand closer or print a bigger marker—rather than relying on candidate retries.

Our study has three primary limitations. First, our data and scope are limited: all conclusions depend on samples from a single synthesizer, which overweights hard cases compared to real-world scenarios. We also have only 12 low-density real images, all of which are cases where the full image already decodes. Second, regarding measurement, equivalent module pixel density is an offline stand-in, so it cannot tell you the true sampling density in different parts of a tilted code. Third, regarding deployment, we have not evaluated MGCS end-to-end on a physical robot; we leave this for future work.

5. Conclusion

Low-resolution QR codes usually need several enhancement attempts, often causing unnecessary decodings and wasted computational time. MGCS mitigates this waste by prioritizing candidates with the highest marginal gain. It maintains a pool of 11 candidate configurations, ranked by a simple ratio of newly decoded codes to execution time. An offline greedy algorithm pre-determines the execution sequence, while online

© 2026 The Author(s). Authors retain copyright in their articles.

decoding terminates immediately upon a valid payload return. No retraining of any restoration model is involved, and the method stays runnable on a CPU.

On two synthetic-degraded benchmark sets which were excluded from candidate selection, MGCS decoded more codes than magnification-only retry. For dataset A, the 5-stage and 10-stage configurations achieved 63.95% and 65.50%, respectively—up 2.80 and 4.35 points. Against a fixed multi-scale retry, 5 stages took about 22% less time on average with no significant accuracy change, while 10 stages added 2.30 points but cost more compute per crop. Validation set B showed the same pattern, indicating that the number of stages lets users trade off accuracy against speed.

With a fixed candidate count, marginal-gain ordering outperformed single-stage-efficiency sorting by 6.25 points on set A and 5.95 on set B. This gain arises from picking candidates that have complementary success sets. While reordering a fixed set affected only timing (all orders yielded identical accuracy), candidate selection clearly dictates accuracy in MGCS, whereas execution order mainly reduces online wait time.

MGCS focuses on situations in which the code region is already cropped and no learned restoration model is available. In the severe degradations we tested, once the equivalent module density drops to 1.2 pixels per module or below, plain enhancement still reads very few codes—candidate retries cannot overcome this limitation; acquisition conditions must be changed instead. Although our approach performs reliably well, it is constrained by a single-synthesizer dataset and sparse real-world samples. Validating practical feasibility through physical robot deployments remains a necessary next step. Future work will focus on gathering real-world capture data to investigate how detection errors and hardware variations impact performance, and on building better methods for extreme imaging conditions.

References

[1] SHI L Y, LIU Z J, JIANG R T. Design and implementation of an intelligent sorting robot based on machine vision[J]. Journal of Liaoning University of Science and Technology, 2026, 28(1): 12-16 [in Chinese].

[2] ZHENG R, YUAN K, LI Y. A QR code for indoor localization and navigation of mobile robots[J]. High Technology Letters, 2008, 18(5): 495-500.

[3] FENG W. Research on fast QR code recognition technology for visual localization[D]. Zibo: Shandong University of Technology, 2019 [in Chinese].

[4] ALSULAMI F, JHANJHI N Z. Deep learning framework for barcode localization and decoding using simulated UAV imagery[J]. Scientific Reports, 2026, 16: 399. DOI:10.1038/s41598-025-29720-w.

[5] FU J T, CONG P, SUN Y W. ESRTMDet: robust recognition of industrial 2D barcodes in complex environments[C]//2024 8th International Symposium on Computer Science and Intelligent Control (ISCSIC). Zhengzhou: IEEE, 2024:227-231.

[6] MIAO Q W, CHEN C, HE W Q. A method for QR code detection and recognition in industrial scenes based on the PPYOLOv2 model[P]. China: CN115270839A, 2022-11-01 [in Chinese].

[7] CORPUZ D J, DEL ROSARIO L V, CEMPRON J P, et al. QR code detection with perspective correction and decoding in real-world conditions using deep learning and enhanced image processing[C]//Proceedings of VISIGRAPP 2025, Volume 3: VISAPP. 2025: 685-690.

[8] ZHENG Y, CHEN W. Method and system for QR code decoding based on distorted image correction[P]. China: CN105046256A, 2015-11-11 [in Chinese].

[9] SÖRÖS G, SEMMLER S, HUMAIR L, HILLIGES O. Fast blur removal for wearable QR code scanners[C]//Proceedings of the 2015 ACM International Symposium on Wearable Computers (ISWC '15). Osaka: ACM, 2015: 117-124.

[10] VEZZALI E, VORABBI L, GRANA C, BOLELLI F. Mosaic-SR: an adaptive multi-step super-resolution method for low-resolution 2D barcodes[C]//IEEE International Conference on Image Processing (ICIP). 2025.

© 2026 The Author(s). Authors retain copyright in their articles.

[11] SANCAR Y. Reconstructing unreadable QR codes: a deep learning based super resolution strategy[J]. PeerJ Computer Science, 2025, 11: e2841.

[12] KEYS R G. Cubic convolution interpolation for digital image processing[J]. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1981, 29(6): 1153-1162.

[13] HENDRYCKS D, DIETTERICH T. Benchmarking neural network robustness to common corruptions and perturbations[C]//International Conference on Learning Representations (ICLR). 2019.

[14] ISO/IEC 18004:2024. Information technology — Automatic identification and data capture techniques — QR Code bar code symbology specification[S]. Geneva: ISO, 2024.

[15] EFRON B. Bootstrap methods: another look at the jackknife[J]. The Annals of Statistics, 1979, 7(1): 1-26.

[16] zxing-cpp: ZXing-C++ barcode decoding library[EB/OL]. https://github.com/zxing-cpp/zxing-cpp.

[17] RICE J R. The algorithm selection problem[J]. Advances in Computers, 1976, 15: 65-118.

[18] VIOLA P, JONES M. Rapid object detection using a boosted cascade of simple features[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2001: 511-518.

Funding

This research received no external funding.

Conflicts of Interest

The author(s) declare no conflicts of interest.

Acknowledgment

This paper is an output of the science project.

Copyrights

Copyright for this article is retained by the author(s), with first publication rights granted to the journal. This is an open-access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/4.0/).

© 2026 The Author(s). Authors retain copyright in their articles.