Table 3: Candidate depth levels for each CTU type.

Types	$Depth_{best\_pred}$	Depth Range [$Depth_{min}$, $Depth_{max}$]
Type 0	$Depth_{best\_pred} = 0$	[0 , 0]
Type 1	$0 < Depth_{best\_pred} \leq 0.5$	[0 ,1]
Type 2	$0.5 < Depth_{best\_pred} \leq 1.5$	[0 , 2]
Type 3	$1.5 < Depth_{best\_pred} \leq 2.5$	[1 , 3]
Type 4	$Depth_{best\_pred} > 2.5$	[2 , 3]

Based on the above analysis, the candidate depth levels that will be tested using RDO for each CTU are summarized in Table 3. Therefore, the dynamic depth level range of CTU of the current CTU can be predicted by using the correlation weight and maximum depth levels of its neighboring blocks. The flowchart of proposed fast encoding method for H.265/HEVC encoder implemented in HM 8.1 [15] is shown in Figure 8. The proposed algorithm firstly utilizes 9 temporally co-located neighboring CTUs to predicted depth level of current CTU. And then, it follows to calculate the predicted depth level value using 4 spatial neighboring CTUs. Finally, we determine the best predicted depth range of current CTU from the intersection of two predicted depth range obtained from tempo-spatial adjacent CTU.

4. Simulation Results

For the performance evaluation, we assess the total execution time of the proposed method in comparison to those of the HM 8.1 [14-15] and the ACUDRD in order to confirm the reduction in computational complexity. The coding performance is evaluated based on ΔBitrate, ΔPSNR and ΔTime, which are defined as follows:

$$\Delta Bitrate = \frac{Bitrate_{method} - Bitrate_{HM8.1}}{Bitrate_{HM8.1}} \times 100\% \quad (5)$$

Table 4: Test conditions and software reference configurations.

Test sequences	• Class A (2560×1600): Traffic • Class B (1920×1080): Kimono, ParkScene, Cactus, BasketballDrive, BQTerrace • Class C (832×480): BasketballDrill, BQMall, PartyScene
Total frames	100 frames
QP	22, 27, 32 and 37
Software	HM 8.1
Scenario Type	Main profile (MP) Low delay P (IPPP...), Random access (IBBB...)

$$\Delta PSNR = PSNR_{method} - PSNR_{HM8.1} \quad (6)$$

$$\Delta Time = \frac{TIME_{HM8.1} - TIME_{method}}{TIME_{HM8.1}} \times 100\% \quad (7)$$

The system hardware is Intel (R) Core(TW) CPU i7-3350P @ 3.40 GHz, 8.0 GB memory, and Window XP 64-bit O/S. Additional details of the encoding environment are described in Table 4.

To evaluate the performance of the proposed method, we incorporated it into test model HM8.1 and implemented it using test sequences recommended by JCT-VC in different resolutions including 2560×1600 (Traffic), 1920×1080 (Kimono, ParkScene, Cactus, BasketballDrive, BQTerrace) and 832×480 (BasketballDrill, BQMall, PartyScene). For a fair comparison, we also implemented the ACUDRD algorithm in [8] in same encoding environment. Table 5 and Table 6 tabulate the performances obtained by testing the HM8.1, the ACUDRD and the proposed method with QP=32, separately. As shown in the Table 5, in a high-efficiency setting with the random access (RA) and the low delay (LD) scenarios, the CTU processing time of proposed method can achieve TIR about 55.73% for RA and 65.77% for LD on average, respectively, as compared to the HM8.1 with insignificant loss of rate distortion (RD) performance which includes bitrate and image quality. On the other hand, Table 6 also shows the performances between the ACUDRD and the proposed method with the same scenario. We also can find that our method can further achieve an average of TIR to 18.33% and 12.31% with RA and LD scenarios, respectively, as compared with the ACUDRD algorithm. It is clear that the proposed fast encoding method indeed efficiently reduce the computational complexity in CU module with insignificant loss of RD performance.